TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Artur Kurvits 185997IABB

ÄRIPROTSESSIDE HALDAMINE MIKROTEENUSTE JA PROTSESSIMOOTORI NÄITEL

Bakalaureusetöö

Juhendaja: Viljam Puusep
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Artur Kurvits

18.05.2022

Annotatsioon

Antud töö uurib võimalusi, kuidas organisatsioon saab hallata äriprotsesse kasutades nii mikroteenuseid, kui ka protsessimootorit. Samuti valmib töö käigus rakendus, mis kasutab mikroteenuste põhimõtteid ning kuhu on ühte teenusesse integreeritud Camunda protsessimootor.

Töö esimene osa kirjeldab mikroteenuste arhitektuuri teoreetilist poolt, annab ülevaate selle plussidest ja miinustest ning võrdleb seda monoliit-arhitektuuriga, mille pealt paljud ettevõtted mikroteenustele üle lähevad. Samuti käsitletakse teemasid nagu mikroteenuste skaleerimine ja teenustevaheline suhtlemine ning tuuakse välja näiteid tehnoloogiatest, millega antud probleeme lahendatakse.

Teine osa tutvustab protsessimootorit, mis see üldse on ning miks seda tihti mikroteenustes kasutatakse. Samuti selgitatakse välja, mida tähendab protsessimootori integreerimine nii arendaja, kliendi, kui ka lõppkasutaja jaoks.

Töö viimases osas kirjeldatakse arendatava tarkvara disainimist ja implementeerimist. Rakenduse kasutajaliides ehitatakse Angulari ja mikroteenuste komponendid Java Spring Boot raamistikus. Teenustevaheliseks suhtlemiseks kasutatakse RabbitMQ-d ning andmebaasisüsteemideks saab PostgreSQL. Kõik moodulid pakitakse kokku Dockeri konteineriteks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 5 peatükki, 36 joonist, 4 tabelit.

Abstract

Managing Business Processes with Microservices and Process Engine

The aim of this bachelor's thesis is to provide options to manage organization's business processes using both microservices and process engine. Also a software, that uses principles of microservices and consists of service that has Camunda process engine integrated in it, will be developed during the process.

The first part of the work describes the theoretical side of microservice architecture, gives an overview of its pros and cons, and compares it with the monolithic architecture from which many companies switch to microservices. Topics such as scaling of microservices and communication between services will also be addressed, as well as technologies that are used to solve these kinds of problems.

The second part introduces what a process engine is and why is it often used in microservices. It also explains what process engine integration means for the developer, the customer, and the end user.

The last part of the thesis describes the design and implementation of the developed software. The user interface of the application will be built in Angular and microservice components in Java Spring Boot framework. RabbitMQ is used for communication between services and the database systems are PostgreSQL. All modules will be built for executable Docker containers.

The thesis is written in Estonian and contains 30 pages of text, 5 chapters, 36 figures, 4 tables.

Lühendite ja mõistete sõnastik

REST	*Representational State Transfer*
URI	*Uniform Resource Identifier*
SOA	*Service Oriented Architecture*
HTTP	*Hypertext Transfer Protocol*
API	*Application Programming Interface*
AMQP	*Advanced Message Queuing Protocol*
SOAP	*Simple Object Access Protocol*
BPM	*Business Process Management*
WFM	*Workflow Management*
WFMS	*Workflow Management System*
ERP	*Enterprice Resource Planning*
JWT	*JSON Web Token*
CI/CD	*Continuous Integration / Continuous Deployment*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeva tarkvaraarenduses käib pea igat uut projekti alustades läbi mõiste „mikroteenused“. Tihti nimetatakse seda infotehnoloogia maailmas uudseks ja innovaatiliseks lähenemiseks, kuid nüüdseks on see teemaks olnud juba mitmeid aastaid ning seda on rakendatud paljudes rakendustes. Tegemist on tehnoloogiaga, millele toetutakse ehitades kvaliteetset ja kergesti skaleeritavat tarkvara.

Käesoleva bakalaureusetöö eesmärgiks on uurida, miks on mikroteenuste arhitektuur muutunud järjest enam populaarseks ning kuidas aitab selle kasutamine kaasa ettevõtte äriprotsessidele. Eesmärgi saavutamiseks selgitatakse välja, kust mikroteenused alguse said ja võrreldakse seda monoliitse arhitektuuriga. Tuuakse välja kuidas on muutunud lähtekoodi ülesehitus ning kuidas see muudab ettevõtte tööd efektiivsemaks. Samuti uuritakse, mis rolli mängib mikroteenustes protsessimootor. Seejärel arendatakse Tervise ja Heaolu Infosüsteemide Keskuse riigihanke [1] proovitööna tarkvara, mis kasutab parimaid mikroteenuste mustreid.

Töö esimene osa keskendub mikroteenuste arhitektuuri teoreetilisele poolele ning võrdleb seda monoliitse arhitektuuriga. Tehakse selgeks mida mikroteenused üldse endast kujutavad, kust see alguse sai ning mis on selle lähenemise eelised ja puudused. Tuuakse välja milliseid protsesse mikroteenuste kasutamine organisatsiooni siseselt lihtsustab, milliseid raskendab. Teises osas keskendutakse äriprotsesside haldamisele - mida see täpsemalt tähendab nii ajalooliselt, kui tänapäeval. Tehakse selgeks, kuidas on see seotud infosüsteemidega ja mis ülesannet täidab protsessimootor tarkvaras üldiselt ja spetsiifiliselt mikroteenustes. Viimases osas kirjeldatakse mikroteenustel põhineva tarkvara disainimist ja implementeerimist programmeerimiskeeles Java [2] ning ühte moodulisse protsessimootori Camunda [3] integreerimist.

1.1 Metoodika

Töö käigus on eesmärk luua töötav ja täisfunktsionaalne tarkvara, mis järgib mikroteenuste parimaid printsiipe ning mille ärilist protsessi juhib Camunda protsessimootor. Töö teostamiseks on autor kasutanud mustreid peamiselt järgmistest teostest: Sam Newmani „Building Microservices“ ja W. M. P. van der Aalst "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow management". Samuti on eesmärgiks mikroteenuseid testida nii ühik-, kui ka integratsioonitestidega. Lõpptulemiks on mikroteenuste arhitektuuril põhinev tarkvara koos kasutajaliidesega ning mille kõik moodulid töötavad Dockeri [4] konteinerites. Arusaadavalt on vaja eesmärkide saavutamiseks palju eksperimenteerida ning õppida kasutama uusi tehnoloogiaid.

1.2 Ülesandepüstitus

Töö eesmärgiks on luua tarkvara, mis vastab Tervise ja Heaolu Infosüsteemide Keskuse korraldatava riigihanke proovitöö nõuetele. Sellest tulenevalt on ülesanne üsna hästi lahti kirjeldatud. Samas on ette antud ka rida nõudeid, mida tuleb ülesande lahendamisel järgida. Vastavalt hankedokumentidele [1] on ülesandeks luua lahendus, kus on kolm erinevat rolli kasutajaid: kasutaja, menetleja ja administraator. Eesmärgiks on kasutajal võimaldada saata teade menetlusse ning menetlejal võimalus teha menetluse otsus mingi ette määratud aja jooksul. Lisaks eelnevale tuleb kõikidest ehitatavatest moodulitest teha GitLabi CI/CD [5] skriptide abil käivitatavad Dockeri konteinerid.

1.3 Ülevaade tööst

Bakalaureusetöö koosneb sissejuhatusest, kokkuvõttest ja kolmest sisupeatükist. Sissejuhatuses käsitletakse töö tausta ning püstitatakse eesmärgid, kirjeldatakse kasutatavat metoodikat ja selgitatakse praktilist ülesannet. Esimeses sisupeatükis selgitatakse täpsemalt lahti mida tähendab mikroteenuste arhitektuur, kust see pärineb, teostatakse võrdlus monoliitse arhitektuuriga ning tuuakse välja plussid ja miinused. Teises sisupeatükis kirjeldatakse, mis tähendab äriprotsesside haldamine, kust see mõiste pärineb, kuidas äriprotsesse automatiseerida ning kuidas sobitub antud konteksti mikroteenuste arhitektuur. Seejärel hakatakse disainima ülesandepüstituses kirjeldatud rakenduse arhitektuuri ja pannakse paika kasutatavad tehnoloogiad, võttes aluseks

hankedokumentides toodu. Viimases peatükis keskendutakse rakenduse implementatsiooni kirjeldamisele, tuues seejuures mitmeid koodinäiteid valminud tarkvarast.

2 Mikroteenuste arhitektuur

Mikroteenuste arhitektuur või lihtsalt mikroteenused on mõnes mõttes eriline meetod arendamaks tarkvara, mis keskendub ühe funktsiooniga moodulite ehitamisele, millel on selgelt määratud liidesed ja operatsioonid. Tänu sellele saab rakendust arendada väikeste, teistest sõltumatute, komplektidena, kus igal teenusel on oma kindel eesmärk. Iga moodul on eraldiseisvalt testitav ja paigaldatav. Samuti võivad need olla kirjutatud erinevates programmeerimiskeeltes ja saavad kasutada erinevaid tehnoloogiaid. [6]

Tegemist on hetkel populaarse teemaga tarkvaraarenduse maailmas ning seda loetakse lahenduseks mitemele tänapäevastele probleemidele nagu näiteks skaleeritavus ja suured mahud. Viimaste aastate jooksul on mikroteenuseid oma süsteemis rakendanud mitmed suurettevõtted eesotsas Netflixi ja Amazoniga.

2.1 Ajalugu

Esimesed alged mikroteenustest pärinevad aastast 2005, kui Dr. Peter Rodgers mainis oma esitluses terminit „Mikro-Veebiteenused“ *(Micro-Web-Services)*. Ta kirjeldas tarkvarakomponente kui mikro-veebiteenuseid, millel on igaühel oma sisemine URI liides. Mikro-veebiteenused liidavad kokku REST veebiteenuste ja unix-tüüpi ajastamise ning suhtluse „torude“ kaudu *(Unix pipelines)*, seejuures on põhiline, et kõik on teenusekeskne. [7]

2010-ndate aastate alguses kerkis lõpuks esile termin „mikroteenused“, kui sellega eksperimenteerisid mitmed ettevõtted. Sel ajal hakkas arhitektuur võtma sellist ilmet, nagu me praegu seda mõistame. Samas väidavad paljud, et kasutasid sarnast lähenemist juba palju varem, kuid ei kasutanud seejuures mõistet „mikroteenus“. Sageli küsitakse, kas mikroteenused pole lihtsalt „teenus-orienteeritud arhitektuur“ *(SOA)*, mida kasutati juba üle kümne aasta tagasi. On tõsi, et neil on üsna palju sarnasusi, kuid tegemist pole siiski sama lähenemisega. Probleem on selles, et SOA võib tähendada mitmeid erinevaid asju ning mikroteenuseid peetakse pigem üheks SOA vormiks. [8]

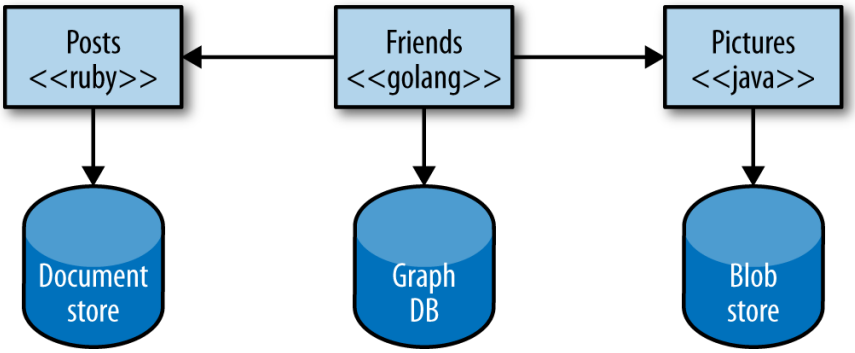
2.2 Monoliit

Mikroteenuste paremaks mõistmiseks on mõistlik seda võrrelda monoliitse arhitektuuriga. Monoliit on sisuliselt rakendus, mis ehitatakse ühe tervikliku tükina. Ettevõtete rakendused on tihti ehitatud kolmest osast: klient-rakendusest (koosneb HTML lehtedest ning JavaScriptist, mis jo [9]okseb kasutaja brauseris), andmebaasist (koosneb mitmetest tabelitest, tavaliselt relatsiooniline andmebaasisüsteem) ning server-rakendusest. Server võtab vastu HTTP päringuid, käivitab äriloogika, loeb ja uuendab andmeid andmebaasis ning koostab HTML vaateid mis saadetakse tagasi brauserisse. Kõik see töötab ühe füüsilise protsessina. [8]

Alustades uue rakenduse arendamist, on algstaadiumis monoliidi kasutamine kõige loogilisem valik, kuna loogikat on veel vähe ning seda tundub sedasi lihtsam hallata. Rakenduse arendamine ning käima panemine on lihtne, kuna tööle tuleb panna vaid üks kokku pakitud fail. Probleemid tekivad siis, kui kood paisub järjest suuremaks ning rakenduse kallal peavad korraga töötama mitmed arendajad. Iga paranduse tegemine nõuab kogu terviku uuendamist ning see on aeganõudev protsess, mis võib nõuda suurt käivitusaega. Samuti on uutel arendajatel tiimiga liituda raskem, kuna koodist arusaamine on keeruline ja väga aeganõudev. Kõige selle tulemina aeglustub kogu arendusprotsess. [6, p. 1]

2.3 Definitsioon

Mikroteenused on väikesed, autonoomsed teenused, mis töötavad omavahel koos. Ühe teenuse eesmärk on teha ainult ühte asja ja teha seda hästi. Tavaliselt tekivad piirid teenuste vahel ärilistest piiridest, et oleks loogiline, milline funktsionaalsus millises koodibaasis elab. Seega on tavaliselt mikroteenuste komponendid jaotatud ärilisteks tükkideks, mis tegelevad näiteks teavituste saatmise, kasutajate haldamise või arvete koostamisega. Seejuures võivad kõik moodulid olla kirjutatud erinevates programmeerimiskeeltes ja kasutatud erinevaid tehnoloogiaid, sest suhtlus käib läbi keelest ja tehnoloogiast sõltumatute API-de. Tänu sellele saab valida parima tööriista just antud probleemi lahendamiseks. [6, p. 1]



Joonis 1. Mikroteenused võivad kasutada erinevaid tehnoloogiaid

Sageli räägitakse teemal, kui suur peaks üks mikroteenus olema. Sõna „mikro“ nagu viitaks sellele, et üks teenus peaks olema hästi pisike. Tegelikkuses see seda üks-ühele siiski ei tähenda. Vaadates ettevõtteid, kes mikroteenuseid kasutavad, leiab nende arsenalist teenuseid, mida haldab kuueliikmeline meeskond, samas on sama suuri meeskondi, kes haldavad korraga kuute eraldi teenust. [8] Lõppkokkuvõttes oleneb siiski mikroteenuse suurus selles paiknevast äriloogika suurusest ning seda koodiridades või muudes numbrites mõõta ei saa.

On mitmeid erinevaid viise kuidas mikroteenused omavahel suhtlevad, igaühel oma stsenaarium, kus ta on kõige kasulikum. Suhtlustüübid saab jagada kaheks.

Esimene defineerib, kas kasutatav protokoll on sünkroonne või asünkroonne:

- Sünkroonne protokoll, näiteks HTTP. Klient saadab päringu ning jääb ootama serveri vastust. Rakendus saab tööd jätkata ainult siis, kui saab kätte vastuse.
- Asünkroonne protokoll, näiteks AMQP. Klientrakendus ei jää pärast päringu saatmist vastust ootama. Saadab lihtsalt sõnumi teele, tavaliselt sõnumite vahendaja *(message broker)* järjekorda.

Teine defineerib, kas saadetaval sõnumil on üks või mitu vastuvõtjat:

- Üks vastuvõtja. Igal päringul peab olema täpselt üks vastuvõtja.
- Mitu vastuvõtjat. Igal päringul võib olla null kuni mitu vastuvõtjat. Selline kommunikatsioonitüüp peab olema asünkronne. Ka sellel juhul kasutatakse tavaliselt sõnumite vahendajat *(message broker)*, sest seal saab lihtsa vaevaga määrata mitu teenust ühte konkreetset järjekorda kuulab.

Mikroteenustel põhinev rakendus kasutab tihti kõiki antud võimalusi kombineeritult vastavalt ärilistele vajadustele. Siiski kõike levinum on sünkroonne päring ühe vastuvõtjaga. [10]

Veel on mikroteenuste puhul oluline, et neil oleks nõrk omavaheline sõltuvus. See tähendab, et üks teenus ei peaks mitte midagi teadma, milline on teise teenuse loogika ning kuidas ta andmeid käsitleb. Oluline on vaid teada, milline on mikroteenuse andmete kuju sisend ja väljund.

2.4 Eelised

Nagu ka varem mainitud, on mikroteenuste suurimaks eeliseks see, et iga äriline funktsionaalsus on eraldiseisvalt käivitatav programm ning iga teenus võib kasutada just endale sobivad tehnoloogiat. Sellest tulenevalt on väiksemate muudatuste tegemine palju efektiivsem, kuna muudatuse sisse viimiseks ei ole vaja maha võtta kogu rakendust ning see uuesti üles panna. Piisab ainult konkreetse mooduli uuesti paigaldamisest, mis toimub kiiresti ning segab ülejäänud rakenduse tööd minimaalselt. Ka vigade tekkimisel on isoleeritud teenuseid lihtsam hallata ja vigu üles leida.

Skaleeritavuse tähendus võib kohati olla erineva tähendusega. See võib olla süsteemi võimekuse tõstmine kasutajate arvu kasvuga. Või see võib tähendada arendusprotsessi võimekust paljudel arendajatel töötada rakendusega paralleelselt. Mikroteenuste puhul on skaleeruv ühik iga teenus. Seega on võimalik suurendada iga teenuse võimekust eraldi, vastavalt vajadusele ja koormusele, võimaldades seejuures kuludelt kokku hoida. [11]

Positiivne mõju on ka arendusprotsessi poole pealt. Suurte rakenduste arendamine käib eraldi meeskondades, mis tähendab, et monoliitse arhitektuuri puhul muudaks sama koodibaasi samal ajal mitmed arendajad erinevatest meeskondadest. See muudab muudatuste haldamise ja kokku panemise keeruliseks. Mikroteenuste puhul saab jagada igale meeskonnale oma teenuse, mille käekäigu eest ta vastutab. Sellisel juhul tuleb koodimuudatusi koordineerida vaid ühe meeskonna siseselt, jättes ülejäänud tiimid sellest välja. [11]

2.5 Puudused

Mikroteenused on hajutatud süsteemid. Hajutatud süsteemide suurim puudus ongi see, et ta on hajus. Kohe kui mängu tuleb hajutatus, kaasneb sellega suur hulk keerukusi. Esimene neist on kiirus. Ehitades monoliiti, ei tule pea kunagi probleemiks rakenduse siseste funktsioonide väljakutsumise kiirus, mikroteenuste puhul on see samas suureks kitsaskohaks. Kui üks päring kutsub välja mitmeid erinevaid teenuseid, mis omakorda teevad päringu veel mitmetesse teenustesse, siis päringuaegade kokkuliitmisel tekib sellest arvestatav ooteaeg. Üks võimalus selle parandamiseks on kirjutada rakendus selliselt, et peaks tegema võimalikult vähe päringuid. See jällegi komplitseerib rakenduse koodi. Teine võimalus on kasutada asünkroonnsust, sellisel juhul on kogupäring sama aeglane, kui kõige aeglasem asünkroonne päring. Seda on jällegi väga keeruline korrektselt tööle saada ning veel keerulisem sellest vajadusel vigu üles leida *(debug)*. [12]

Lisaks viiteaja probleemile, lisandub mikroteenustega ka veahalduse keerukus. Iga päringuga peab arvestama, et see võib ebaõnnestuda ning alati ei pruugi olla üheselt selge, millisest teenusest viga pärines.

Kuna mikroteenuseid on tavaliselt palju, siis on üheks probleemiks ka nende infrastruktuuri haldamine. Kui monoliidi puhul pole vajab paigaldamist vaid üks rakendus ning sellega saab hakkama ka üks inimene manuaalselt, siis mikroteenustega on asi keerulisem. Mitmete ja mitmete teenuste opereerimine ja manuaalselt paigaldamine on praktiliselt võimatu, see tähendab, et vajalik on eraldi meeskond, kes tegeleks teenuste infrastruktuuri ning paigalduste automatiseerimisega. [12]

Mikroteenuste juures räägitakse ka probleemidest testimisega. Keerukus antud juhul tekib teenuste omavahelistest sõltuvustest. Suuremate testide jaoks tuleb käivitada ka testitavast teenusest sõltuvad teenused. Teine võimalus on kirjutada teste selliselt, et sõltuvad moodulid on kontekstist sõltuvalt konfigureeritavad või tehakse testide jaoks nendele võltsingud *(mock)*.

3 Äriprotsesside haldamine

Mida aeg edasi, seda rohkem kogub populaarsust ettevõtetes distsiplineeritud äriprotsesside juhtimine. Sellest räägitakse paljudel konverentsidel ning kirjutatakse mitmeid artikleid. Kuid mida see täpsemalt siiski tähendab, kuidas see muudab suurte ettevõtete tööd paremaks ning kuidas sobitub antud konteksti mikroteenuste arhitektuur?

3.1 Ajalugu

Alates esimesest tööstusrevolutsioonist on produktiivsus märgatavalt kasvanud, seda peamiselt erinevate tehniliste innovatsioonide ja infotehnoloogia tõttu. Esimese tööstusrevolutsiooni (1784-1870) ajal võeti kasutusele vee ja aurujõul töötavad masinad. Teine revolutsioon (1870-1969) baseerus masstootmisel ning elektrienergia kasutamisel, kolmas (1969-2015) arvutite, võrkude ja muude IT süsteemide kasutamisel. Tänapäeval räägitakse uuest revolutsioonist „Industry 4.0“, mis tähendab „tarkade“ tootmissüsteemide loomist kasutades erinevaid sardsüsteeme, sensoreid, võrke, suurandmeid *(big data)* ja analüütikat. [13]

Kuigi eelnevalt väljatoodud tööstusrevolutsioone seostatakse tavaliselt tehaste ja füüsiliste tootmistega, siis tegelikult käib see kõik samamoodi administratiivsete protsesside ja teenuste kohta. Arvutite turule tulek muutis radikaalselt ettevõtete tööd ja tootmisprotsesse. Sellistes moodsates protsessides on tooteks tihti informatsioon, mis on edastatud teenuse kaudu, mitte enam nii palju füüsiline ese. Antud protsesside automatiseerimiseks võeti kasutusele töövoo haldussüsteem *(WFM)*. [13]

3.2 Töövoo haldus(süsteemid)

Töövoo haldamise koalitsioon *(Workflow Management Coalition)* defineerib töövoogu: „Äriprotsessi täielik või osaline automatiseerimine, mille käigus dokumendid, teave või ülesanded antakse ühelt osalejalt teisele tegutsemiseks üle vastavalt protseduurireeglitele“ [14]. Töövoo haldussüsteem on defineeritud: „Süsteem, mis määratleb, loob ja haldab töövoogude täitmist, kasutades tarkvara, mis töötab ühel või mitmel protsessimootoril, mis suudab protsessi tõlgendada, määratleda, suhelda töövoos osalejatega ja vajadusel kasutada IT tööriistu ja rakendusi“ [14]. Mõlema puhul tuleb

välja automatiseerimine ja süsteemi kasutamine, ehk näiteks sellise tarkvara kasutamine, mis toetab protsesside elluviimist.

On selge, et töövoo haldus on asjakohane iga organisatsiooni jaoks, kuid reaalsuses on üsna vähe organisatsioone, kes kasutaksid „päris“ töövoo haldussüsteemi. WFM süsteemid jaotatakse nelja kategooriasse: puhtad WFM süsteemid *(pure WFM systems)*, WFM komponendid, mis on liidetud teiste süsteemidega, spetsiaalselt valmistatud *(custom-made)* WFM lahendused ja rakendustesse sisse kodeeritud *(hard-coded)* WFM lahendused. Aastal 2004 kirjutati, et tollel ajal kasutasid enamus organisatsioonid endiselt kahe viimase kategooria lahendusi. [15]

3.3 Definitsioon

Äriprotsesside haldus *(BPM)* on distsipliin, mis kombineerib teadmisi infotehnoloogiast ja juhtimisteadustest ning rakendab neid operatiivsetele äriprotsessidele. See on oluline osa ettevõtte eesmärkide saavutamiseks läbi protsessi optimeerimise ja eesmärgistatud juhtimise. Põhjalik äriprotsesside juhtimine muudab ettevõtet efektiivsemaks, aidates seeläbi tõsta käivet, hoida kokku kulusid ning suurendada ka kliendirahulolu. Tehes mingit protsessi efektiivselt tähendab, et sama aja ja tööjõu kuluga on võimalik teenida suuremat tulu ja samal ajal kasvatada lõpptarbija rahulolu.

BPM sisaldab meetodeid, tehnikaid ja tööriistu, mis aitavad protsesse disainida, manageerida ja analüüsida. BPM-i elutsükkel koosneb neljast etapist:

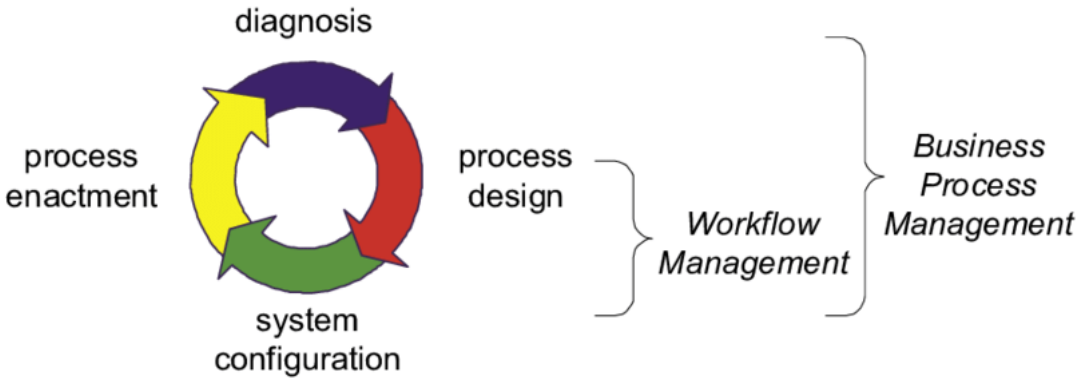
- Protsessi disain *(Process design)*. Iga BPM algab praeguse *(as-is)* või soovitava *(to-be)* protsessi modelleerimisega.

- Süsteemi konfigureerimine *(System configuration)*. Protsessi disaini põhjal hakatakse realiseerima protsessi-teadlikku *(process-aware)* infosüsteemi. Traditsiooniliselt tähendab see aeganõudvat tarkvaraarendus protsessi. Teine võimalus on kasutada juba valmis tarkvara, sellisel juhul tarkvaraarendus asendub konfigureerimise ning komplekteerimise protsessiga.

- Protsessi jõustumine *(Process enactment)*. Etapp, kus eelmises etapis realiseeritud infosüsteem võetakse päriselt kasutusele.

- Hindamine *(Diagnosis)*. Et protsessi-teadlik infosüsteem areneks edasi, tuleb seda aja jooksul optimeerida, katsetada uusi tehnoloogiaid, toetada uusi protsesse ning kohandada pidevalt muutuva keskkonnaga. See tähendab, et hindamise etapp ühendab protsessi jõustumise etapi uue disaini etapiga.

Nagu tavaliselt tarkvaraarenduse tsüklites, on ka BPM-i neli elutsüklit kattuvad ja kogu protsess on korduv. [15]



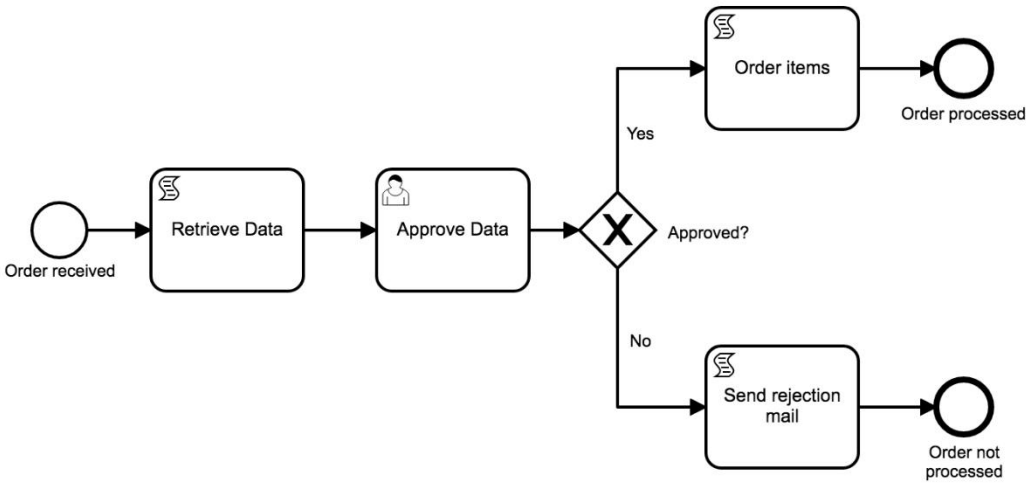
Joonis 2. BPM elutsükkel

BPM-i võib vaadata kui WFM-i kontseptsiooni edasiarengut. WFM keskendub peamiselt äriprotsesside automatiseerimisele, BPM aga on laiem mõiste. See tähendab, et tegeletakse alates protsessi automatiseerimisest ja analüüsist kuni operatsioonide juhtimise ja töökorralduseni. Ühest küljest on eesmärgiks parandada äriprotsesse võimalusel ilma uute tehnoloogiateta. Teisest küljest on BPM siiski sageli seostatud tarkvaraga, millega juhitakse, kontrollitakse ja teotatakse tööprotsesse. Sellest sai alguse uut tüüpi tehnoloogia, mida kutsutakse BPM süsteemideks. Tegemist on süsteemidega, mis saavad ühenduda väga paljude erinevate rakendustega ning on edukalt välja vahetanud nende eelkäijad, WFM süsteemid. [13]

3.4 Protsessimudel

Protsessimudeli mõiste on BPM-i aluseks, selle eesmärgiks on ära kaardistada kõik võimalikud moodused, kuidas juhtumit *(case)* saab käsitleda. Äriprotsesside mudeli loomiseks on olemas hulgaliselt tähistussüsteeme, näiteks BPMN, Petri nets, UML. Kõigi ühiseks tunnuseks on see, et protsessid on kirjeldatud tegevustena ja alamtegevustena. Samuti võib protsessimudel kirjeldada erinevaid ajalisi omadusi ja andmete kasutust,

näiteks otsuste modelleerimiseks või näitamaks, kuidas protsessid mingeid ressursse kasutavad. [16]



Joonis 3. Näidis BPMN protsessimudel

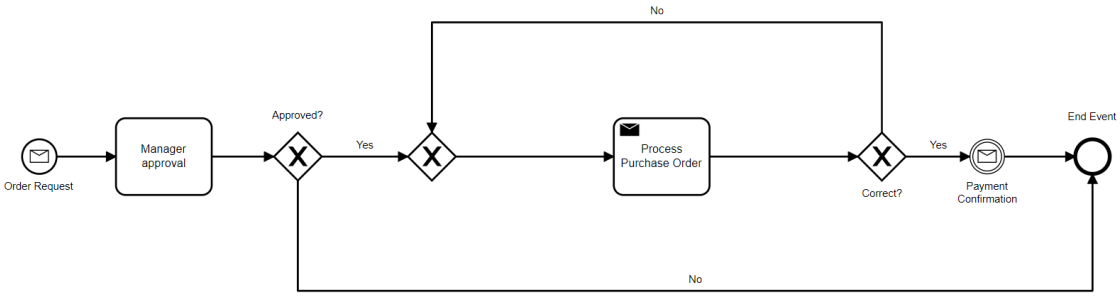
Joonisel 3 on toodud näidiseks lihtne tellimuse töötlemise protsessimudel BPMN notatsioonis. Protsess algab uue tellimuse vastuvõtmisest *(order received)*, seejärel päritakse tellimuse andmed *(retrieve data)* ning suunatakse töötajale, kes peab otsustama, kas andmed on korrektsed või mitte *(approve data)*. Andmete kinnitamisel võetakse kõik tellitud tooted *(order items)* ning kinnitatakse tellimus *(order processed)*. Vastasel juhul saadetakse kliendile e-kiri tellimuse tagasilükkamise kohta *(send rejection mail)* ning tellimus lükatakse tagasi *(order not processed)*.

3.5 Protsessimootor ja mikroteenused

Lihtsustatult on protsessimootori (*process engine,* erinevas kirjanduses kasutatud ka termineid nagu *workflow engine* ja *execution engine)* puhul tegemist süsteemiga, mis hoolitseb töövoo koordineerimise, kontrollimise ja täideviimise eest [16]. Protsessimootor töötab tavaliselt koos vastava teenusega tema käitusajal *(runtime)*, töötades keskse agendina, mis kontrollib protsesside orkestreerimist [17]. Tegelikult on paljudesse ERP süsteemidesse protsessimootori komponendid juba sisse ehitatud, kuid sageli antud lahendused ei sobi ning ehitades nullist täiesti uut tarkvaralahendust, tekib seal vajadus protsessimootori järele.

Kuna huvi BPMN 2.0 protsessimudelite vastu kasvab, siis on välja tulnud uus generatsioon avatud lähtekoodiga protsessimootoritest. Nende eesmärk on käivitada eeldefineeritud BPMN 2.0 protsessimudel, kus on selgelt määratud, kuhu ja mis järjekorras mingi informatsioon jõudma peab. Samal ajal on protsessimootor alati teadlik, kus etapis protsess hetkel on ning mis on järgmine samm. Selliseid süsteeme on võimalik integreerida programmikoodi ning seeläbi võimaldab käivitada erinevaid tarkvaraülesandeid ja protsesse välistes süsteemides. Lisaks suudavad kaasaegsed BPM süsteemid defineeritud protsesse ka monitoorida, administreerida ja optimeerida [18].

Tulles nüüd mikroteenuste juurde, siis sisuliselt käib protsessimootori integreerimine sama moodi nagu ükskõik millisesse rakendusse. Vahe seisneb selles, et mikroteenustes vastutab iga teenus ühe äriprotsessi eest, seega võib protsessimootori instantse olla mitu. See jällegi tähendab, et protsessimootori saab lisada vaid nendesse moodulitesse, kus see on ka päriselt vajalik. [19] Näiteks võib tuua e-poe, mis koosneb kolmest mikroteenuse moodulist: kasutajad, tellimused ja maksmine. Kasutajate ja maksmise mooduli puhul pole protsessimootor otseselt vajalik. Kasutajate teenus haldab sisselogimist ja kasutajate infot ning maksmise moodul salvestab makseinfot ning suhtleb pankadega. Need on üsna lihtsad ülesanded ning protsessimootor suurt eesmärki seal ei täidaks. Räägites aga tellimuste moodulist, siis sinna võiks protsessimootori integreerida, kuna seal on vaja erinevate osapoolte kinnitusi ning vastavalt nende otsustele võib protsess erinevalt käituda. Joonisel 4 on toodud näidis protsessimudel tellimuse töötlemiseks.



Joonis 4. Tellimuse protsessimudel

Nagu varem mainitud, siis mikroteenus peab olema täiesti eraldiseisev ja teistest sõltumatu moodul. Kui varasemalt pakkusid protsessimootorid eraldi platvormi, et ehitada üles terve protsessidele orienteeritud rakendus, siis viimastel aastatel turule tulnud protsessimootoreid, nagu näiteks Activiti [20] või Camunda [3], saab käivitada Java teegina mikroteenuse mooduli sees. Sellisel juhul ei ole selle jaoks vaja ka eraldi serverit,

virtuaalmasinat või isegi Dockeri konteinerit, kõik jookseb koos ühes Java virtuaalmasinas *(Java Virtual Machine)*. Seega on integreerimine programmikoodi sedavõrd lihtsam ning jääb alles ka mikroteenuse isoleeritus, sõltumatus ja võimalus igal moodulil iseseisvalt suuremaks kasvada.

4 Rakenduse disain

Töö käigus lõi autor täisfunktsionaalse tarkvara koos Gitlabi pipeline skriptide ja käivitatavate Dockeri [4] konteineritega. Rakendus koosneb API gateway, mikroteenuste, sõnumivahetuse ning esitlus kihtidest. Samuti on ühte mikroteenusesse integreeritud Camunda BPM vastava protsessi juhtimiseks. Rakendus sai tehtud riigihanke [1] proovitööna, seega ülesande püstitus tuleneb sealt.

4.1 Arhitektuur

Nagu iga tarkvaralahendusega, algas ka antud rakenduse ehitamine esmalt selle kavandamisest. Kuna tegemist pidi olema mikroteenus-arhitektuuril põhineva rakendusega, lisas see veel keerukust juurde. Ülesandeks oli koostada lihtsakoeline menetlus-haldus süsteem, kus on kolm kasutajarolli.

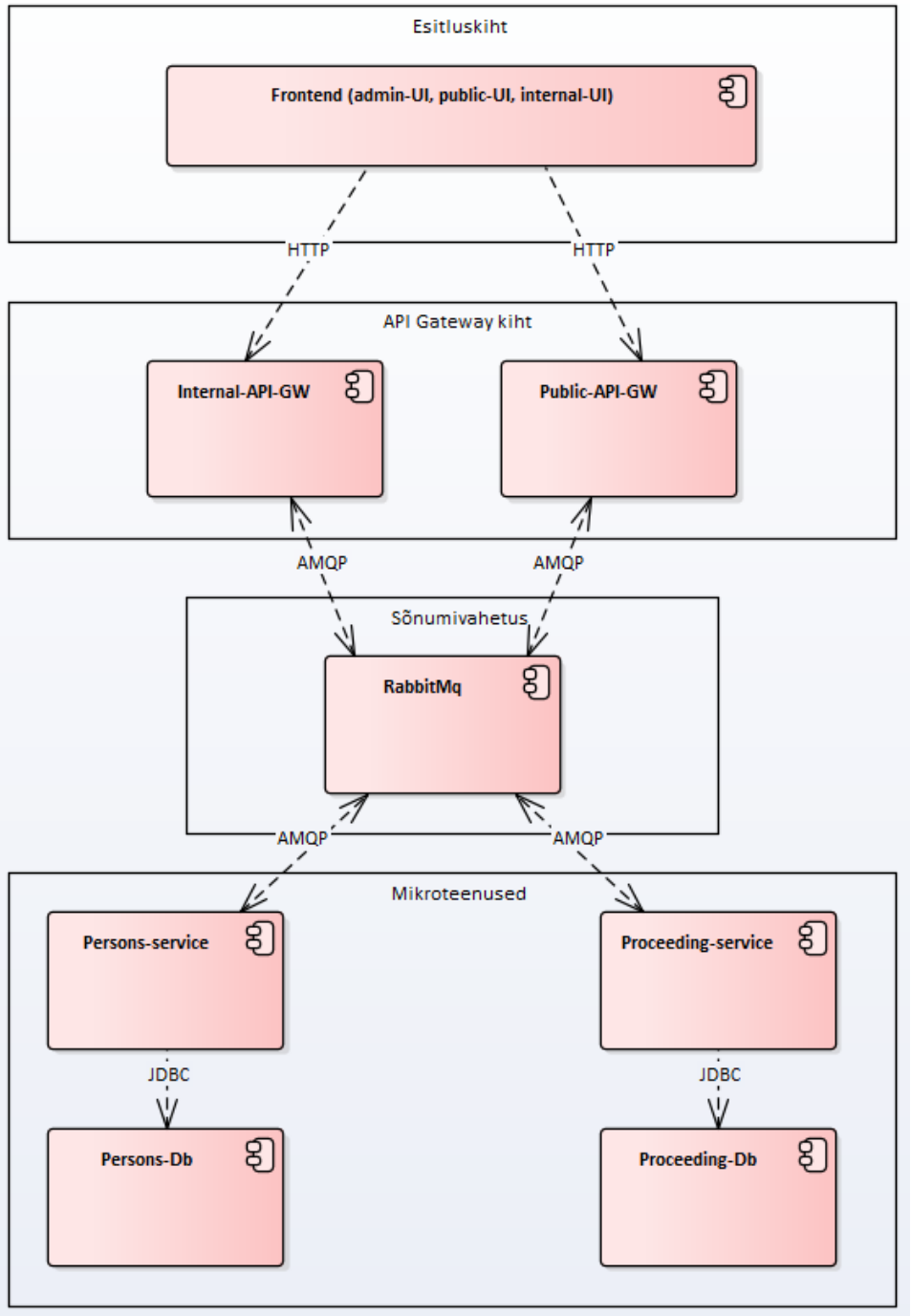
- **Administraator** saab aktsepteeritud identsustõendi kaudu lisada süsteemi uusi kasutajaid.
- **Kasutaja** saab koostada teate ning saata selle menetlusse.
- **Menetleja** saab menetluses olevale teatele kirjutada vastussõnumi ning selle kas aktsepteerida või tagasi lükata. Kui etteantud aja jooksul menetlusele ei vastata, loetakse see automaatselt aegunuks.

Antud kasutajarolle ja ülesandeid arvesse võttes jaotati kavandatav rakendus järgmisteks komponentideks:

- **Esitluskiht** ehk *front-end* sisaldab igale rollile vastavat kasutajaliidest.
- **API Gateway kiht**, ehk tööriist, mis on esitluskihi ning äriloogikat sisaldavate mikroteenus moodulite vahel. See jaguneb kaheks mooduliks: sisene *(internal)* ja avalik *(public)*. Läbi sisese gateway liigub administraatorite ja menetlejate päringud, olles toodangupaigalduses välismaailmale suletud. Avalik gateway on kõikidele võrkudele avatud ning läbi selle saavad rakendust kasutada tavakasutajad.

- **Sõnumivahetus -** keskne teenus, mis vahendab AMQP sõnumeid gateway ja mikroteenus moodulite vahel.

- **Mikroteenused –** koosneb kahest moodulist: kasutajate teenus *(persons-service)* ja menetlusteenus (*proceeding-service).* Nendele lisandub kummagi teenuse kohta oma andmebaas.



Joonis 5. Komponentdiagramm

4.2 Tehnoloogiad

Mikroteenuste puhul võivad teenused kasutada erinevaid tehnoloogiaid, vajalik on vaid ühine sõnumivahetus loogika. Näidisrakenduses on siiski kasutatud sarnaseid tehnoloogiaid keerukuse vältimiseks ning lihtsamaks haldamiseks.

4.2.1 Programmeerimiskeeled

Kõik server-teenused on kirjutatud kasutades Java 11-s, mis on üheks populaarsemaks programmeerimiskeeleks ning sobib mikroteenuste konteksti hästi. Kompileerimiseks ning teekide haldamiseks on kasutusel Gradle [21]. Esitluskihis on kasutusel HTML 5, CSS ja Typescript 4.4.4.

4.2.2 Raamistikud/teegid

Kõikides *back-end* moodulites on kasutusel Spring Boot 2.5.6 [22] ja Hibernate. Sõnumivahetusteenusega suhtlemiseks on kasutusel Spring AMQP ja andmebaasioperatsioonideks Liquibase [23] ja Spring Data JPA. Autentimiseks ning tokenite genereerimiseks JJWT [24], konteinerite ehitamiseks Jib [25]. *Front-end* raamistikuks on Angular 13 [26].

4.2.3 Andmebaasisüsteemid

Andmebaasisüsteemides on kasutusel PostgreSQL 13.4, kuna see on väga populaarne ning kõrgelt arenenud avatud lähtekoodiga *(open-source)* relatsiooniline andmebaas. Lisaks kõigele muule, on Postgres tuntud kui väga töökindel, heade funktsioonidega ja paljude laiendusvõimalustega andmebaas. [27]

4.2.4 Konteinerid

Rakenduse arhitektuur on üles ehitatud konteinerite peale. Selleks kasutatakse Dockeri [4] konteinereid, kus igast teenusest ehitatakse valmis Dockeri kujutis *(image)* Java teenuste puhul kasutades Jib pluginat ning *front-end*-i puhul kasutades konfiguratsioonifaili *Dockerfile*. Siiski on võimalik kõiki mooduleid käivitada ka konteineriväliselt.

4.2.5 Muu tarkvara

Sõnumitevahetus teenuseks on kasutusel RabbitMQ [28], mille kasutamine oli ülesande kirjelduses ette antud. Samas rakendus toimiks sarnaselt ka kasutades mõnda muud AMQP protokolli pakkuvat teenust.

Ülesande püstitusest tulenevalt tuli protsessi juhtimiseks kasutada Camunda BPM [3] lahendust. Kuna põhiline äriprotsess antud näidisrakenduses toimub *proceeding-service* moodulis, siis tuli sellesse integreerida Camunda.

Koodi halduseks kasutati GIT-i, samuti loodi rakendusele GitLabi CI/CD [5] skriptid, mis käivitasid automaattestid, kompileerisid rakenduse ning ehitasid Dockeri *image*.

5 Implementatsioon

Käesolevas peatükis on detailselt lahti kirjutatud arendatud tarkvara tehnoloogilised lahendused ja andmebaaside struktuurid. Samuti on toodud mitmeid koodinäiteid ja kasutajaliidese erinevad vaated.

5.1 API Gateway kiht

Nagu varasemalt mainitud, siis API Gateway kiht koosneb avalikust *(public)* ja sisemisest *(internal)* moodulist. Selle kihi eesmärk on võtta vastu *front-end*-i poolt saadetavad HTTP päringud, need muuta AMQP jaoks sobivasse formaati ning saata üle RabbitMQ vastavasse mikroteenusesse, vea ilmnemisel (või RabbitMQ-st vastuse mitte saamisel) püütakse see *ExceptionHandler*-ga kinni ning saadetakse korrektne HTTP staatuskood koos veateatega kasutajaliidesesse. Samuti on peal ka JWT filter ning Spring Security mis kontrollib, kas antud kasutajal on ligipääs vastavale REST API-le või mitte.

5.1.1 Public-api gateway

Läbi avaliku gateway toimub kasutajate autentimine ning seejärel tavakasutaja suhtlus mikroteenustega. Samuti luuakse sisselogimisel kasutajaandmete põhjal JWT token, mille abil saab kasutajaliides edaspidi suhelda.

Tabel 1. Public-api gateway HTTP liidesed

HTTP meetod	URL	Kirjeldus
GET	/api/authenticate	Pärib kasutaja andmed *person-service*-st ning genereerib nende põhjal tokeni, millega logitakse kasutaja sisse, parameeter: • *username* – otsitav kasutajanimi
GET	/api/proceeding	Tagastab kõik sisselogitud kasutaja menetlused.
POST	/api/proceeding/add	Loob uue menetluse, *body* parameetrid:

		• *subject* – menetlusteate pealkiri • *body* - menetlusteate sisu
POST	/api/proceeding/read	Märgib vastuse saanud menetluse kasutaja poolt loetuks, *body* parameeter: • *id* – vastava menetluse UUID
POST	/api/proceeding/resend	Märgib aegunud menetluse kasutaja poolt vaadatuks ning vajadusel saadab uuesti menetlusse, *body* parameetrid: • *id* – vastava menetluse UUID • *resendRequired* – kas kasutaja soovib teadet uuesti menetlusse saata – *true* või *false*

5.1.2 Internal-api gateway

Sisese gateway kaudu toimub administraatorite ja menetlejate suhtlus mikroteenustega, tavakasutajatel ligipääs puudub. Autentimist ja autoriseerimist teostatakse sisselogimisel loodud JWT tokeni abil.

Tabel 2. Internal-api gateway HTTP liidesed

HTTP meetod	URL	Kirjeldus
POST	/api/person/add	Võimaldab administraatoril lisada identsustõendi alusel uue kasutaja, *body* parameeter: • *token* – uue kasutaja identsustõend
GET	/api/person/all	Võimaldab administraatoril kasutajate haldamiseks pärida list

		kõikidest olemasolevatest kasutajatest.
GET	/api/proceeding/processing	Võimaldab menetlejal pärida kõik menetluses olevad teated.
POST	/api/proceeding/answer	Võimaldab menetlejal menetluses olevale teatele vastata, *body* parameetrid: • *id* – vastava menetluse UUID • *status* – uus menetluse staatus, tagasi lükatud või aktepteeritud. • *answer* - vastussõnum

5.1.3 Koodinäited

Lisas 2 on välja toodud mõningad koodinäited gateway moodulitest. Esimesena on näide internal-api gateway Spring Security konfiguratsioonist, kus on määratud igale HTTP päringule filter. See kontrollib, et igal sisse tulnud päringul oleks kaasas header JWT tokeniga. Järgmiseks on konfiguratsioonis määratud, mis kasutajarollid millistele API-dele ligi pääsevad. See on oluline selleks, et vale rolliga kasutaja ei saaks teha tegevusi, mille jaoks tal õigusi ei ole.

Järgmiseks on näide ühest REST API-st. Tegemist on public-api gateway *„/api/authenticate“* API-ga, mis võtab kasutaja päringu vastu ning koostab sellest sõnumi RabbitMQ jaoks sobilikule kujule. Pärast vastuse saatmist genereeritakse unikaalne JWT token ning tagastatakse see kasutajale. Antud REST API on standartne Spring Boot lahendus, seega kõik API-d käesolevas rakenduses on analoogsed.

Viimaseks on koodinäide, kus saadetakse eelnevalt kontrollerist vastu võetud ning RabbitMQ jaoks sobivale kujule teisendatud sõnum edasi RabbitMQ järjekorda. Selle

jaoks on Spring raamistikus olemas spetsiaalne RabbitTemplate klass, millele on varasemalt konfiguratsiooniparameetritena ette antud RabbitMQ aadress. Edasi saab selle klassi kaudu saata sõnum järjekorda, määrates ära võtme (antud näites *„api.getPerson“*), mille abil RabbitMQ teab, millisesse järjekorda sõnum läheb. Pärast vastuse saamist kontrollitakse selles errorite olemasolu ning nende puudumisel tagastatakse vastussõnum.

5.2 Kasutajate teenus

Kasutajate teenuse *(persons-service)* eesmärk on säilitada kasutajate andmeid ning neid vastavalt päringutele kasutajanime järgi väljastada. Teenus võtab vastu päringuid RabbitMQ järjekordadest ja tagastab andmeid kasutajate andmebaasist. Andmebaasi struktuurimuudatuste haldamiseks on kasutusel Liquibase ja andmeobjektide käsitlemiseks ning andmebaasipäringute teostamiseks kasutatakse Spring Data JPA teeki. Tulenevalt rakenduse lihtsusest sisaldab kasutajate andmebaas vaid ühte tabelit, kuid sellise lahenduse juures on seda võimalik väga lihtsalt laiendada. Samuti on teenuse ülesandeks käivitamisel luua kõik kasutajate teenuse jaoks vajalikud RabbitMQ järjekorrad, kui neid veel ei eksisteeri.

Tabel 3. Kasutajate teenuse andmemudel

Tabeli nimetus	Väärtuse nimetus	Väärtuse kirjeldus
person	id	UUID tüüpi primaarvõti
	username	Kasutajanimi
	role	Kasutaja roll, võimalikud väärtused: • USER • PROCEDER • ADMIN

	token	Administraatori poolt sisestatud kasutaja identsustõend

5.2.1 Koodinäited

Iga mikroteenuse moodul loob enda jaoks vajalikud RabbitMQ järjekorrad. Lisas 3 on toodud näide järjekordade loomise konfiguratsioonist. Igal rakenduse käivitamisel kontrollitakse üle, kas vajalikud järjekorrad juba on olemas. Puuduste olemasolul järjekorrad luuakse vastavalt konfiguratsioonile. Lisaks on välja toodud näide RabbitMQ järjekorra kuulamisest. Selle jaoks on kasutusel Spring AMQP standartne lahendus, kus defineeritakse meetod koos järjekorra võtmega, mida kuulama hakatakse. Meetodi parameetriks on klass, mis sisaldab vastuvõetava sõnumi struktuuri.

Lisas 4 on kasutajate teenuse klassid, mis esitavad andmebaasiobjekti Java klassina. *AbstractEntity* klass on vaikimisi klass, kus on defineeritud primaarvõti, mis on kasutamiseks kõikidele objektidele. Käesolevas lahenduses on primaarvõtme tüübiks UUID, kuna see on inimese jaoks loetamatul kujul ning seega on raskem päringuid manipuleerida. Samuti ei ole UUID võtmed loogiliselt järjestatavad, erinevalt tavalisest arv-võtmest. *PersonEntity* klass on *person* tabeli andmete lugemiseks/kirjutamiseks.

Lisas 5 on toodud näide *person* repositooriumist ning andmekirje salvestamisest *person* tabelisse. Kasutades Spring JPA repositooriumit, on koheselt olemas lihtsamad operatsioonid nagu näiteks kirje salvestamine ja kõikide kirjete pärimine. Lisaks saab vajadusel juurde kirjutada spetsiifilisemaid päringuid, antud juhul näiteks kasutaja pärimine kasutajanime (*findByUsername)* või kasutajate pärimine rolli järgi (*findByRole)*. Andmekirje salvestamiseks *person* tabelisse tuleb selleks luua *PersonEntity* objekt, väärtustada vajalikud väljad ning see repositooriumi abil salvestada.

Andmebaasi struktuuri eest vastutab Liquibase, mis loob kasutatavasse andmebaasi tabeli *databasechangelog*, kus hoitakse infot, millised skriptid ja muudatuste komplektid on juba andmebaasi jõudnud ja millised mitte. Lisas 6 on toodud komplekt kasutajate mooduli andmebaasi struktuuri loomiseks. Sama lahendust kasutab ka menetluste moodul, seega on kõik klassid selles analoogsed.

5.3 Menetlusteenus

Menetlusteenuse *(proceeding-service)* ülesandeks on uue menetlusteate loomisel käivitada Camunda protsess menetluse juhtimiseks ning säilitada oma andmebaasis menetluste andmeid. Sarnaselt kasutajate teenusele on andmebaasi haldamiseks kasutusel Liquibase ja Spring Data JPA ning teenus loob käivitamisel kõik vajalikud RabbitMQ järjekorrad. Lisaks menetluse tabelile on samas andmebaasis ka Camunda teenuse jaoks vajalikud tabelid.

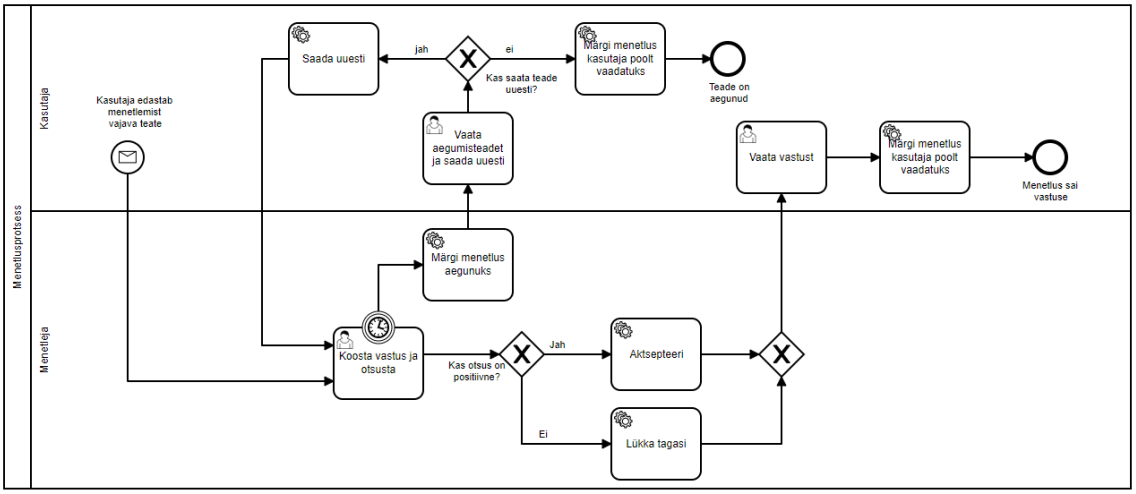
Tabel 4. Menetlusteenuse andmemudel

Tabeli nimetus	Väärtuse nimetus	Väärtuse kirjeldus
proceeding	id	UUID tüüpi primaarvõti
	sender	Teate saatja kasutajanimi
	subject	Teate pealkiri
	body	Teate sisu
	send_timestamp	Teate saatmise kuupäev ja kellaaeg
	status	Teate staatus, võimalikud väärtused: • PROCESSING • APPROVED • REJECTED • EXPIRED
	answer	Vastussõnum
	proceder	Vastaja menetleja, kasutajanimi

	answer_timestamp	Vastuse saatmise kuupäev ja kellaaeg
	deadline	Vastuse aegumise kuupäev ja kellaaeg
	answer_read_timestamp	Kasutaja vastuse lugemise kuupäev ja kellaaeg

5.3.1 Protsess

Et menetlemise mikroteenust üldse arendama hakata, tuli esmalt luua protsessimudel, mida see peaks järgima hakkama. Selleks oli hea kasutada Camunda Modeleri, kus oli võimalik lihtsalt vajalik mudel BPMN 2.0 notatsioonis valmis joonistada, vajalikud parameetrid määrata, salvestada .bpmn failina ning lisada programmi ressursside hulka.



Joonis 6. Menetluse protsessimudel

Protsess läheb käima hetkel, kui kasutaja loob uue teate ning saadab selle menetlusse. Seejärel jääb teade vastuse ootele, kui täitub aeg, mil menetlus aegub, siis muudetakse teate staatus ning saadetakse tagasi kasutajale. Kui kasutaja on aegunud teadet lugenud, on tal valik, kas saata teade uuesti menetlusse (sel juhul jõuab teade uuesti vastuse ootamise etappi) või mitte (protsess lõppeb). Juhul, kui menetleja teatele vastab, muudetakse teate staatust vastavalt menetleja otsusele ning jääb ootama kasutaja vastuse lugemist. Kui kasutaja on vastuse ära lugenud, muudetakse menetlus loetuks ning protsess lõppeb.

5.3.2 Camunda integreerimine

Camunda lisamine Spring Boot rakendusse on võrdlemisi lihtne. Nagu varem mainitud, siis esmalt tuli valmis teha protsessimudel ning sinna määrata igale ülesandele *(task)* vastav nimi. Seejärel saab Java koodist vastavaid taske käivitada. Protsessi algatamisel luuakse Camunda sisene *taskId*, mis käesolevas rakenduses seotakse konkreetse menetluse ID-ga ning mille kaudu saab järgmise etapi käivitamiseks õigele protsessile viidata. Samuti saavad Camunda *task*-id käivitada ka Java koodi. Näiteks teate aegumisel käivitatakse kood, mis muudab vastava teate staatust ning salvestab selle andmebaasi.

5.3.3 Koodinäited

Lisas 7 on toodud erinevaid koodinäiteid Camunda integreerimisest. Esimesena on välja toodud meetod, mis alustab Camunda protsessi. Selleks määratakse protsessi ID (milleks on menetluse andmebaasi UUID koos eesliitega, näiteks *APPROVAL_ f3e84750-bb7b-42e2-a4b9-8a401fca7c7d)* ning aeg, kui kaua on menetlejal aega menetlusotsus teha. Seejärel käivitatakse protsessiinstants, kuhu antakse kaasa eelnevalt määratud muutujad.

Järgmiseks on meetod, mille abil saab leida käimasoleva protsessi identifikaatori, et kasutajate tehtud otsused ühendada Camunda protsessiga. See leitakse üles menetluse UUID ja vastava ülesande (*taski*) nime järgi. Kui protsessi üles ei leita on järelikult protsess katki ning kasutajale tagastatakse viga.

Seejärel on näiteks toodud meetod, mis käivitatakse, kui menetleja on langetanud otsuse. Menetluse ID järgi otsitakse käimasolev protsess ja sellel väärtustatakse vastav muutuja menetluse otsusega. Seejärel jätkatakse Camunda protsessi, mis siis käitub vastavalt tehtud otsusele.

Viimaseks on näide Java koodi käivitamisest Camunda protsessi poolt. Tegemist on tegevusega, mis läheb käiku, kui menetleja ette määratud aja jooksul otsust ei langeta. Protsessist saadakse kätte menetluse ID, mille alusel algatatakse loogika menetluse aegumiseks.

Lisas 8 on välja toodud meetod, mis salvestab menetluse otsuse andmebaasi ning saadab otsuse ka Camunda protsessile. Esmalt valideeritakse vastussõnumi olemasolu, seejärel otsitakse andmebaasist menetluse ID põhjal vastav kirje ja kontrollitakse, et menetlus

poleks staatuses, mille ajal ei ole menetlejal õigust otsust langetada. Vigade tekkimisel tagastatakse veasõnum kasutajale. Kui vigasid ei teki, muudetakse *ProceedingEntity* kirjel vajalikud väärtused ning salvestatakse see andmebaasi, saadetakse info Camunda protsessile ning tagastatakse kasutajale äsja salvestatud andmed.

5.4 Testimine

Rakendust programmeerides on mõeldud ka automaattestide peale. Kõikide Java moodulite jaoks on kirjutatud Junit 5 [29] ja Mockito [30] ühiktestid *(unit tests)*. Lisaks on mõlemale mikroteenuse moodulile tehtud ka integratsioonitestid. Kasutades Testcontainers [31] teeki käivitatakse koos testidega RabbitMQ ja PostgreSQL konteinerid, kuhu lisatakse Liquibase abiga testandmed. Integratsioonitestide eesmärk on kontrollida reaalsete ühenduste loomist ning andmebaasipäringuid. Kõikide moodulite testidega katvus on vähemalt 85%.

Ühiktestide puhul on ignoreeritud väljakutsutavaid teenuseid teistes klassides ja keskendutud on sellele, et kontrollitav meetod tagastaks vastavalt sisendile eeldatava väärtuse. Selleks on väliste meetodite kutsumised võltsitud (*mock)* nii, et need tagastaksid erinevaid väärtusi, mida saab eeldada, et nad tagastavad.

Integratsioonitestides käivitatakse konteinerites nii RabbitMQ, kui PostgreSQL ning pannakse rakendus tööle. Seejärel saadetakse järjekorda sõnum. Selle abil saab kontrollida, kas järjekord eksisteerib ning kas tagastatav vastus on korrektne. Samuti saab valideerida, kas salvestatav kirje ka päriselt andmebaasi salvestus ning kas saadi kätte need kirjed, mis konteineri käivitumisel andmebaasi sisestati. Testitakse läbi kõik RabbitMQ järjekorrad. Lisaks on tehtud ka testid, mis kontrollivad Camunda protsessi. Selleks kävitatakse rakendus koos Camundaga, algatatakse seal protsess ning kontrollitakse erinevaid otsuseid ja protsessi olekuid.

5.4.1 Koodinäited

Lisas 9 on toodud kaks näidet ühiktestide kohta kasutajate moodulis. Esimeses testitakse meetodit, mis peab tagastama listi kasutaja menetlustest. Selleks luuakse kaks menetluse

objekti ning Mockito abil määratakse, et andmebaasi päringule tagastataks list antud objektidest. Seejärel kontrollitakse, et meetod tagastas lõpuks oodatud tulemused ning vahepeal ei tekkinud ühtegi viga. Teises näites testitakse RabbitMQ järjekorra kuulaja meetodit, mis peab tagastama kõik rakenduses olevad kasutajad. Seejärel kontrollitakse, et meetodis ei tekkinud ühtegi viga ning tagastati eeldatavad tulemused. Veel testitakse ka seda, et meetod tagastaks õige tulemi, kui kasutajate pärimisel ilmneb viga.

Lisas 10 on esiteks näide integratsioonitestide konfiguratsioonist. Seal on kirjeldatud millised testkonteinerid testi käivitumisel käima lähevad ning määratakse nende konfiguratsioonimuutujad. Tegemist on klassiga, mida peavad pärima testklassid, mis loodud konteinereid testides kasutavad. Teiseks on toodud näide menetlusstaatuses olevate teadete otsimiseks. Selleks koostatakse esmaslt päringu objekt, mis saadetakse vastavasse RabbitMQ järjekorda. Sealt vastuse saamisel kontrollitakse, kas vastuses on vaid nii palju kirjeid, kui testkonteinerite käivitamisel andmebaasi sisestati ning kas päringu töötlemisel tekkis mõni viga.

5.5 Kasutajaliides

Kasutajaliidese poolele sai tehtud kolm erinevat vaadet – igale kasutajarollile, lisaks veel lihtne sisselogimise leht. Lihtsuse huvides toimub sisselogimine vaid kasutajanime alusel. Rakenduse esmakordsel käivitamisel on olemas vaid *admin* kasutaja, kes peab edasisteks toiminguteks lisama juurde uusi kasutajaid. Disain ja muudeks *front-end* funktsionaalsuste arendamiseks on kasutusel Angular Material [32]. Tegemist on Angulari tegijate enda arendatud teegiga, mis võimaldab üsna lihtsalt kasutusele võtta erinevaid komponente. Käesolevas rakenduses on kasutusel näiteks erinevad modal-id (uue teate sisestamine ja teate lugemine) ja tabelid (menetluste, kasutajate kuvamine).

Lisas 11 on näited kõikidest vaadetes, mida erinevad rollid näha saavad. Esimesena on sisselogimise leht, seejärel *admin* rolli vaade ning *kasutaja* rolli tavavaade ning modal uue teate sisestamiseks. Edasi *menetleja* vaade ja modal teatele vastamiseks. Viimaseks *kasutaja* modal sõnumi lugemiseks.

5.6 Konteineriteks ehitamine

Kõigi rakenduse moodulite konteineriteks ehitamine toimub igal *git push* operatsioonil Gitlab-i keskkonnas CI/CD skriptide abil. See on populaarne keskkond koodi ning pideva integreerimise ja tarnimise haldamiseks. Antud implementatsioonis on Gitlabi töövoo tulemiks käivitatavad konteinerid Gitlabi konteinerite registris (*Container registry)*.

Java moodulite konteinerite ehitamiseks kasutatakse Jib teeki ja kuna iga mooduli ehitamine on täpselt samasugune, on selle jaoks tehtud üldised skriptid, mida kasutavad kõik ehitatavad Java moodulid. Lisas 12 on esmalt näiteks võetud skript kasutajate moodulist. Antud skript impordib töövood üldkasutatavatest skriptidest. Edasi on välja toodud töövoole eelnevad tegevused (Gradle asukoha määramine ja versiooninumbri moodustamine), seejärel testimise ning konteineriks ehitamise etapid. Üldiselt on loogika lihtne – käivitatakse tavalised Gradle käsud ning vajadusel ehitatakse eelnevalt sõltuv moodul. Jib teek ehitab konteineri ise valmis ning publitseerib selle muutujatega ette määratud konteinerite registrisse.

Kasutajaliidese ehitamine on Java moodulitest pisut erinev. Et Angulari rakendus konteinerisse saada, tuleb esmalt luua *Dockerfile*, kus on kirjeldatud selle ehitamiseks vajalikud käsud. Seejärel saab Gitlabi CI/CD skripti esimeses etapis kompileeritud rakenduse selle abil Dockeri konteinerisse ning publitseerida konteinerite registrisse. Lisaks on loodud võimalus kasutada erinevat *npm* repositooriumit vajalike pakettide installimiseks. Lisas 13 on nii *Dockerfile*, kui Gitlabi skript.

5.7 Edasiarendused

Valminud rakendus on küll täisfunktsionaalne, kuid siiski minimaalsete funktsioonidega. Sellegipoolest kuna rakenduse arhitektuuris on järgitud parimaid mikroteenuste mustreid, siis on uute mikroteenuse moodulite ja olemasolevate täiendamine tehtud võimalikult lihtsaks. Konkreetsemalt rääkides on esimese asjana mõistlik ilmselt teha korrektne kasutajate haldamise süsteem. Kasutades selleks siis mõnda välist autentimisvahendit või arendada rakendusesisene (näiteks klassikaline kasutajanimi-parool lahendus).

Üheks suuremaks puuduvaks komponendiks täislahendusest on korralik konteinerite haldussüsteem, mille abil saaks teha rakenduse kasutajatele kättesaadavaks ning millega saaks edasisi tarneid automaatselt toodangusse saata (*deploy*-da). Üheks võimalikuks (ja väga laialt levinud) valikuks oleks Kubernetes [33]. Selle üheks suureks eeliseks on hea integratsioon Gitlab-ga, mis võimaldab teha automaatseid paigaldusi ka CI/CD töövoos. Veel edasi minnes saaks teha sarnaselt üldistele CI/CD skriptidele ka Helm [34] kaardid, mis muudaks paigaldused veelgi lihtsamaks ning võimaldaks mikroteenuseid konfigureerida vastavalt konkreetse infosüsteemi vajadustele.

Kokkuvõte

Töö eesmärgiks oli luua täisfunktsionaalne tarkvara, mis vastaks Tervise ja Heaolu Infosüsteemide Keskuse hanke proovitöö nõuetele. Lisaks sellele oli eesmärgiks uurida mikroteenuste arhitektuuri ja äriprotsesside haldamist protsessimootori abil ning antud teadmisi ja parimaid mustreid kasutada rakenduse arendamisel. Arendusprotsess sisaldas endas palju katsetamist ja eksperimenteerimist erinevate tehnoloogiatega, kuna enamused nendest olid töö autorile uued.

Esmalt kirjeldas töö autor mikroteenuste teoreetilist poolt ja kust sellline lähenemine alguse sai. Samuti toodi välja mikroteenuste eelised ja puudused ning teostati põgus võrdlus monoliitse arhitektuuriga. Tuli välja, et mikroteenustele sarnane lähenemine ei ole enam väga uus, seda nimetatakse tihti hoopis teenus-orienteeritud arhitektuuri liigiks. Siiski on mikroteenused kogunud viimaste aastate jooksul palju populaarsust ja see on põhjusega – eriti mahukate infosüsteemide puhul.

Teises osas uuris autor äriprotsesside haldamise kohta. Kust see mõiste pärineb ja mida täpsemalt tähendab. Veel uuriti protsessimootori kohta ning kuidas see sobitub mikroteenuste konteksti. Tuli välja, et sellel on tõesti oma koht infosüsteemides ja seega ka mikroteenustes. Üheks põhjuseks on ka uued lahendused, mille integreerimine rakenduse koodi on tehtud üsna lihtsaks.

Kolmandas osas kasutas autor õpitud teadmisi ja mustreid selleks, et disainida arhitektuur ja määrata täpsed tehnoloogiad hanke proovitöö lahenduse implementeerimiseks. Toodi välja arhitektuuripilt ning kõik kasutatavad tehnoloogiad erinevate probleemide lahendamiseks. Seejärel kirjeldati rakenduse teostust koos koodinäidete ja selgitustega, kuidas ja mis põhjustel probleemid selliselt lahendatud said.

Tulemuseks on töötav ja täisfunktsionaalne tarkvara, mis järgib mikroteenuste arhitektuuri mustreid ning mille ühte moodulisse on edukalt integreeritud Camunda protsessimootor. Rakendus töötab täiesti hajutatult ja kõik moodulid on ehitatud Dockeri konteineritesse, olles valmis koheselt sobivasse keskkonda paigaldamiseks ja töö alustamiseks. Samuti sai hea tulemuse rakendus ka riigihanke hindajatelt, seega võib lugeda töö alguses püstitatud eesmärgid edukalt läbituks.

Kasutatud kirjandus

[1] Tervise ja Heaolu Infosüsteemide Keskus, "Terviseameti menetlussüsteemi arendus- ja hooldustööd (MEIS)," 27 December 2021. [Online]. Available: https://riigihanked.riik.ee/rhr-web/#/procurement/3967748/general-info. [Accessed 19 April 2022].

[2] Oracle, "Java Software," Oracle, [Online]. Available: https://www.oracle.com/java/. [Accessed 12 April 2022].

[3] Camunda, "The Universal Process Orchestrator," Camunda, [Online]. Available: https://camunda.com/. [Accessed 12 April 2022].

[4] Docker, "docker," [Online]. Available: https://www.docker.com/. [Accessed 19 April 2022].

[5] "GitLab CI/CD," GitLab, [Online]. Available: https://docs.gitlab.com/ee/ci/. [Accessed 20 April 2022].

[6] Smartbear.com, "What are Microservices?," [Online]. Available: https://smartbear.com/solutions/microservices/. [Accessed 12 April 2022].

[7] K. D. Foote, "A Bried History of Microservices," 22 April 2021. [Online]. Available: https://www.dataversity.net/a-brief-history-of-microservices/. [Accessed 12 April 2022].

[8] M. Fowler, "Microservices," 25 March 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed 12 April 2022].

[9] S. Newman, Building Microservices, 2015.

[10] Microsoft, "Communication in a microservice architecture," 15 September 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture. [Accessed 13 April 2022].

[11] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Software*, vol. 35, 2018.

[12] M. Fowler, "Microservice Trade-Offs," 1 July 2015. [Online]. Available: https://martinfowler.com/articles/microservice-trade-offs.html. [Accessed 13 April 2022].

[13] W. M. P. van der Aalst, M. La Rosa and F. M. Santoro, "Business Process Management," p. 6, 4 January 2016.

[14] P. Lawrence, Workflow handbook, New York: John Wiley & Sons, Inc., 1997.

[15] W. M. P. van der Aalst, "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow management," in *Lectures on Concurrency and Petri Nets*, 2004, pp. 1-66.

[16] W. M. P. v. d. Aalst, "Business Process Management: A Comprehensive Survey," 12 February 2013.

[17] M. Weske, Business Process Management: Consepts, Languages, Architectures, 2007.

[18] A. Delgado, D. Calegari, P. Milanese, R. Falcon and E. Garcfa, "A Systematic Approach for Evaluating BPM Systems: Case Studies on Open Source and Proprietary Tools," in *Open Source Systems: Adoption and Impact*, 2015, pp. 81-90.

[19] A. M. Guti´errez–Fern´andez, M. Resinas and A. Ruiz–Cort´es, "Redefining a Process Engine as a Microservice Platform," in *Business Process Management Workshops*, Rio de Janeiro, 2016, pp. 252-263.

[20] Activiti, "Open Source Business Automation," [Online]. Available: https://www.activiti.org/. [Accessed 19 April 2022].

[21] "Gradle Build Tool," Gradle Inc, [Online]. Available: https://gradle.org/. [Accessed 20 April 2022].

[22] "Spring Boot," VMware, [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 20 April 2022].

[23] "Liquibase - Open Source Version Control for Your Database," [Online]. Available: https://www.liquibase.org/. [Accessed 20 April 2022].

[24] "Java JWT: JSON Web Token for Java and Android," [Online]. Available: https://github.com/jwtk/jjwt. [Accessed 20 April 2022].

[25] "Jib," [Online]. Available: https://github.com/GoogleContainerTools/jib. [Accessed 20 April 2022].

[26] "Angular," Google, [Online]. Available: https://angular.io/. [Accessed 20 April 2022].

[27] PostgreSQL, "PostgreSQL: The World's Most Advanced Open Source Relational Database," [Online]. Available: https://www.postgresql.org/. [Accessed 19 April 2022].

[28] RabbitMQ, "RabbitMQ," [Online]. Available: https://www.rabbitmq.com/. [Accessed 19 April 2022].

[29] "JUnit 5," [Online]. Available: https://junit.org/junit5/. [Accessed 21 April 2022].

[30] "mockito," [Online]. Available: https://site.mockito.org/. [Accessed 21 April 2022].

[31] "Testcontainers," [Online]. Available: https://www.testcontainers.org/. [Accessed 21 April 2022].

[32] "Angular Material," Google, [Online]. Available: https://material.angular.io/. [Accessed 14 May 2022].

[33] "Kubernetes," [Online]. Available: https://kubernetes.io/. [Accessed 14 May 2022].

[34] "Helm The package manager for Kubernetes," [Online]. Available: https://helm.sh/. [Accessed 14 May 2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Artur Kurvits

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Äriprotsesside haldamine mikroteenuste ja protesssimootori näitel“, mille juhendaja on Viljam Puusep.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Gateway koodinäited

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
            .addFilter(new JwtRequestFilter(authenticationManager()))
            .authorizeRequests()
            .antMatchers( ...antPatterns: "/api/person/add", "/api/person/all")
                .hasAuthority("ADMIN")
            .antMatchers( ...antPatterns: "/api/proceeding/processing", "/api/proceeding/answer")
                .hasAuthority("PROCEDER")
            .anyRequest().authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

Joonis 7. Internal-api gateway Spring Security konfiguratsioon

```
@GetMapping(value = "/api/authenticate")
public ResponseEntity<String> getPerson(@RequestParam String username)
        throws AuthenticationServiceException {
    log.info("Finding user with username: " + username);
    PersonRequest request = new PersonRequest();
    request.setPayload(new PersonRequest.Parameters(username));
    PersonResponse response = personProcessingService.getPerson(request);

    log.info( message: "Generating jwt token and authenticating user: {}",
            response.getPayload().getUsername());
    String token = jwtBuilder.generateTokenAndAuthenticate(response.getPayload());
    return ResponseEntity.ok(token);
}
```

Joonis 8. Autentimise HTTP liides

```
public PersonResponse getPerson(PersonRequest request) {
    PersonResponse response =
            (PersonResponse) rabbitTemplate.convertSendAndReceive( routingKey: "api.getPerson", request);
    if (response != null) {
        if (response.getError() != null) {
            throw new MeisRestException(response.getError());
        }
        log.info( message: "Received person from rabbit: {}", response.getPayload());
        return response;
    }
    log.error("Error reaching persons module");
    throw new NoResponseFromRabbitException(COMPONENT_NAME);
}
```

Joonis 9. Sõnumi saatmine RabbitMQ-sse

Lisa 3 – RabbitMQ järjekordade loomine ja kuulamine

```java
@Component
public class QueueConfig {
    private final AmqpAdmin amqpAdmin;

    public static final String GW_GET_PERSON_QUEUE = "persons-service.gw.getPersonRequest";
    public static final String GW_GET_ALL_PERSONS_QUEUE = "persons-service.gw.getAllPersonsRequest";
    public static final String GW_ADD_PERSON_QUEUE = "persons-service.gw.addPersonRequest";

    public static final String GW_INTERNAL_EXCHANGE = "meis.rabbit.gw-exchange-internal";
    public static final String GW_PUBLIC_EXCHANGE = "meis.rabbit.gw-exchange-public";

    public QueueConfig(AmqpAdmin amqpAdmin) { this.amqpAdmin = amqpAdmin; }

    @PostConstruct
    public void createQueues() {
        Queue getPersonQ = new Queue(GW_GET_PERSON_QUEUE);
        Queue addPersonQ = new Queue(GW_ADD_PERSON_QUEUE);
        Queue getAllPersonsQ = new Queue(GW_GET_ALL_PERSONS_QUEUE);
        amqpAdmin.declareQueue(getPersonQ);
        amqpAdmin.declareQueue(addPersonQ);
        amqpAdmin.declareQueue(getAllPersonsQ);

        DirectExchange internalEx = new DirectExchange(GW_INTERNAL_EXCHANGE);
        DirectExchange publicEx = new DirectExchange(GW_PUBLIC_EXCHANGE);
        amqpAdmin.declareExchange(internalEx);
        amqpAdmin.declareExchange(publicEx);

        Binding addPersonInternal = BindingBuilder.bind(addPersonQ).to(internalEx)
                .with( routingKey: "api.addPerson");
        Binding getAllPersonsInternal = BindingBuilder.bind(getAllPersonsQ).to(internalEx)
                .with( routingKey: "api.getAllPersons");
        Binding getPersonPublic = BindingBuilder.bind(getPersonQ).to(publicEx)
                .with( routingKey: "api.getPerson");
        amqpAdmin.declareBinding(addPersonInternal);
        amqpAdmin.declareBinding(getAllPersonsInternal);
        amqpAdmin.declareBinding(getPersonPublic);
    }
}
```

Joonis 10. RabbitMQ järjekordade loomine

```java
@RabbitListener(queues = {QueueConfig.GW_ADD_PERSON_QUEUE})
public PersonResponse addPerson(AddPersonRequest request) {
    PersonResponse response = new PersonResponse();
    response.setUser(request.getUser());
    try {
        logMessageReceived( method: "addPerson", request.getUser().getUsername());
        PersonDTO dto = personService.addPerson(request);
        response.setPayload(dto);
    } catch (Exception ex) {
        logErrorCatched( method: "addPerson", request.getUser().getUsername(), ex.getMessage()
        response.setError(new ErrorDTO(ex.getMessage()));
    }
    return response;
}
```

Joonis 11. RabbitMQ järjekorra kuulamine

Lisa 4 – Kasutajate teenuse andmebaasiobjekti klassid

```java
@Data
@MappedSuperclass
@NoArgsConstructor
public abstract class AbstractEntity {
    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    private UUID id;
}
```

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
@Table(name = "person")
public class PersonEntity extends AbstractEntity {
    @NonNull
    private String username;

    @NonNull
    private String role;

    @NonNull
    private String token;
}
```

Joonis 12. Andmebaasiobjekti klassid

Lisa 5 – Kasutaja salvestamine andmebaasi

```java
public interface PersonRepository extends JpaRepository<PersonEntity, UUID> {
    @Transactional
    PersonEntity findByUsername(String username);

    @Transactional
    List<PersonEntity> findByRole(String role);
}
```

```java
public PersonDTO addPerson(AddPersonRequest request) {
    String token = request.getPayload().getToken();
    DecodedToken decodedToken = decodeToken(token);
    if (decodedToken == null) {
        log.error("Invalid token: " + request.getPayload().getToken());
        throw new IllegalArgumentException(ErrorMessages.INVALID_TOKEN);
    }

    PersonEntity existingPerson = personRepository.findByUsername(decodedToken.getUser());

    if (existingPerson != null) {
        throw new IllegalArgumentException(ErrorMessages.USER_EXISTS);
    }

    PersonEntity personEntity = new PersonEntity();
    personEntity.setUsername(decodedToken.getUser());
    personEntity.setRole(mapUserRoleFromToken(decodedToken.getRole()));
    personEntity.setToken(token);
    personRepository.save(personEntity);

    return buildPersonDTO(personEntity);
}
```

Joonis 13. Kasutaja salvestamine andmebaasi

Lisa 6 – Liquibase konfiguratsioon kasutajate andmebaasi struktuuri loomiseks

```xml
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-2

    <changeSet id="2021-11-11_final" author="artur.kurvits@industry62.com">
        <!--
            Caused by: org.postgresql.util.PSQLException: ERROR: function uuid_generate_v4() does not exist
            https://debugah.com/pgsql-error-function-uuid_generate_v4-does-not-exist-how-to-solve-17244/
        -->
        <sql endDelimiter=";">
            CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
        </sql>

        <createTable tableName="person">
            <column name="id" type="uuid" defaultValueComputed="uuid_generate_v4()">
                <constraints primaryKey="true" primaryKeyName="pk_person" unique="true"/>
            </column>
            <column name="username" type="varchar(100)">
                <constraints nullable="false" unique="true"/>
            </column>
            <column name="role" type="varchar(50)">
                <constraints nullable="false"/>
            </column>
            <column name="token" type="clob">
                <constraints nullable="false"/>
            </column>
        </createTable>

        <sql endDelimiter=";">
            ALTER TABLE person ADD CONSTRAINT check_role CHECK (role IN ('USER', 'PROCEDER', 'ADMIN'));
        </sql>
    </changeSet>

</databaseChangeLog>
```

Joonis 14. Liquibase skript

Lisa 7 – Camunda protsesside integreerimine

```
public void startProcess(String proceedingId) {
    log.info( message: "___ Start process for proceedingId={}", proceedingId);
    String processBusinessId = getProceedingBusinessId(proceedingId);
    Map<String, Object> processVariables = new HashMap<>();
    processVariables.put(VAR_PROCEEDING_ID, proceedingId);
    processVariables.put(VAR_APPROVAL_WAIT_TIMEOUT, approvalWaitTimeout);


    ProcessInstance processInstance = runtimeService
            .startProcessInstanceByMessage(MSG_PROCESS_BY, processBusinessId, processVariables);
    log.info( message: "___ New process is started with id={}", processInstance.getId());

}
```

Joonis 15. Camunda protsessi algatamine

```
private String findTaskId(String proceedingId, String taskDefinitionKey) {
    String processBusinessId = getProceedingBusinessId(proceedingId);

    TaskEntity taskEntity = (TaskEntity) taskService.createTaskQuery()
            .processInstanceBusinessKey(processBusinessId)
            .taskDefinitionKey(taskDefinitionKey)
            .active().singleResult();
    if (taskEntity == null) {
        log.error( message: "User task {} {} not found", proceedingId, taskDefinitionKey);
        throw new IllegalArgumentException("Otsitavat " + proceedingId + " " + taskDefinitionKey + " tööd ei leitud");
    }
    return taskEntity.getId();
}
```

Joonis 16. Camunda *taskId* leidmine mentluse ID ning käimasoleva *task*-i järgi

```
public void approveRejectTask(String proceedingId, String newStatus) {
    String taskId = findTaskId(proceedingId, TASK_PROCEEDING_APPROVAL_KEY);
    log.info( message: "{} user task {} {} newStatus={}",
            TASK_PROCEEDING_APPROVAL_KEY, proceedingId, taskId, newStatus);
    taskService.setVariable(taskId, VAR_PROCEEDING_STATUS, newStatus);
    taskService.complete(taskId);
}
```

Joonis 17. Camunda protsessi jätkamine peale vastussõnumi saatmist

```
@Log4j2
@Component
public class ExpireServiceTask implements JavaDelegate {

    @Autowired
    private ProceedingService proceedingService;

    @Override
    public void execute(DelegateExecution delegateExecution) throws Exception {
        String uuidString = (String) delegateExecution.getVariable(ProceedingProcess.VAR_PROCEEDING_ID);
        log.info( message: "___ Expire service task {}", uuidString);

        proceedingService.expireProceedingByTask(uuidString);
    }
}
```

Joonis 18. Java koodi käivitamine teate aegumisel

Lisa 8 – Menetluse otsuse salvestamine andmebaasi

```
public ProceedingDTO answerProceeding(AnswerProceedingRequest request) {
    if (request.getPayload().getAnswer().isEmpty()) {
        throw new IllegalArgumentException(ErrorMessages.INVALID_INPUT);
    }

    Optional<ProceedingEntity> proceeding = proceedingRepository
            .findById(request.getPayload().getId());
    if (proceeding.isPresent()) {
        ProceedingEntity proceedingEntity = proceeding.get();

        if (proceedingEntity.getStatus().equals(ProceedingStatusEnum.EXPIRED.name())) {
            throw new IllegalArgumentException(ErrorMessages.PROCEEDING_EXPIRED);
        } else if (!proceedingEntity.getStatus().equals(ProceedingStatusEnum.PROCESSING.name())) {
            throw new IllegalArgumentException(ErrorMessages.PROCEEDING_STATUS_MISMATCH);
        }

        proceedingEntity.setAnswer(request.getPayload().getAnswer());
        proceedingEntity.setProceder(request.getUser().getUsername());
        proceedingEntity.setAnswerTimestamp(LocalDateTime.now());
        ProceedingEntity savedEntity = proceedingRepository.save(proceedingEntity);
        String newStatus = request.getPayload().getStatus().name();

        proceedingProcess.approveRejectTask(savedEntity.getId().toString(), newStatus);
        return buildDTO(savedEntity);
    }
    throw new IllegalArgumentException(ErrorMessages.PROCEEDING_NOT_FOUND);
}
```

Joonis 19. Menetluse otsuse salvestamine

Lisa 9 – Ühiktestid

```java
@Test
void testGetProceedings() {
    String sender = "kasutaja";
    UUID id1 = UUID.randomUUID();
    ProceedingEntity entity1 = new ProceedingEntity();
    entity1.setId(id1);
    entity1.setStatus(ProceedingStatusEnum.PROCESSING.name());
    entity1.setSendTimestamp(LocalDateTime.now().minusDays(2));
    UUID id2 = UUID.randomUUID();
    ProceedingEntity entity2 = new ProceedingEntity();
    entity2.setId(id2);
    entity2.setStatus(ProceedingStatusEnum.APPROVED.name());
    entity2.setStatus(ProceedingStatusEnum.APPROVED.name());
    entity2.setSendTimestamp(LocalDateTime.now().minusDays(1));

    when(proceedingRepository.findBySender(sender)).thenReturn(List.of(entity1, entity2));

    ProceedingRequest request = new ProceedingRequest();
    request.setUser(new UserDTO(sender, UserRoleEnum.USER.name()));
    List<ProceedingDTO> response = proceedingService.getProceedingsByUser(request);
    assertThat(response.size()).isEqualTo(2);
    assertThat(response.get(0).getId()).isEqualTo(id2);
    assertThat(response.get(1).getId()).isEqualTo(id1);
}
```

Joonis 20. Mockito ühiktest kasutaja menetluste pärimiseks

```java
@Test
void testGetAllPersons() {
    when(personService.getAllPersons()).thenReturn(List.of(samplePerson));
    testListenerResponse(personRequest, personGwListener.getAllPersons(personRequest),  isError: false);

    when(personService.getAllPersons()).thenThrow(new IllegalArgumentException("Error"));
    testListenerResponse(personRequest, personGwListener.getAllPersons(personRequest),  isError: true);
}

private <T extends AbstractBaseDTO<PT>, R extends AbstractBaseDTO<PR>, PT, PR> void
    testListenerResponse(T request, R response, boolean isError) {
    if (isError) {
        assertThat(response.getPayload()).isNull();
        assertThat(response.getError()).isNotNull();
    } else {
        assertThat(response.getPayload()).isNotNull();
        assertThat(response.getError()).isNull();
    }
    assertThat(response.getUser()).isEqualTo(request.getUser());
}
```

Joonis 21. Kasutajate RabbitMQ listeneri ühiktest

Lisa 10 - Integratsioonitestid

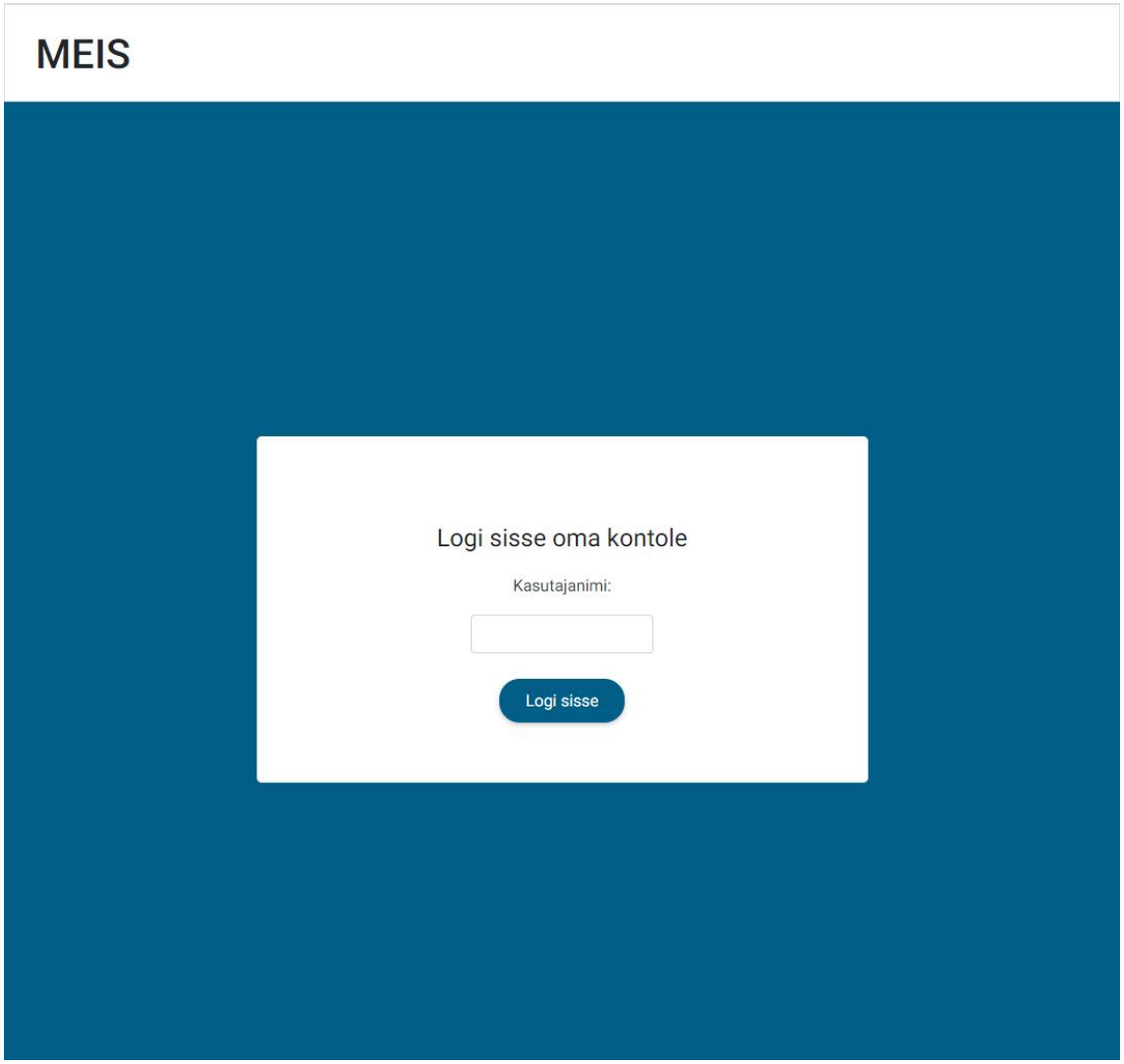
```java
@SpringBootTest(classes = ProceedingApplication.class)
@ActiveProfiles("test")
@Testcontainers
@ContextConfiguration(initializers = {ProceedingServiceAppTestBase.Initializer.class})
public class ProceedingServiceAppTestBase {
    private static final RabbitMQContainer rabbitMQContainer;
    private static final PostgreSQLContainer<?> postgresContainer;

    static {
        rabbitMQContainer = new RabbitMQContainer( dockerImageName: "rabbitmq:3-management")
                .withUser( name: "meis",  password: "meis")
                .withExposedPorts(5672);
        if (!rabbitMQContainer.isRunning()) rabbitMQContainer.start();

        postgresContainer = new PostgreSQLContainer<>( dockerImageName: "postgres:13.4")
                .withDatabaseName("meis-proceeeding")
                .withUsername("meis")
                .withPassword("meis")
                .withExposedPorts(5432);
        if (!postgresContainer.isRunning()) postgresContainer.start();
    }

    static class Initializer implements ApplicationContextInitializer<ConfigurableApplicationContext> {
        @Override
        public void initialize(ConfigurableApplicationContext configurableApplicationContext) {
            TestPropertyValues
                    .of("spring.rabbitmq.port=" + rabbitMQContainer.getFirstMappedPort(),
                            "spring.rabbitmq.host=" + rabbitMQContainer.getContainerIpAddress(),
                            "spring.rabbitmq.username=" + rabbitMQContainer.getAdminUsername(),
                            "spring.rabbitmq.password=" + rabbitMQContainer.getAdminPassword(),
                            "spring.datasource.url=" + postgresContainer.getJdbcUrl(),
                            "spring.datasource.username=" + postgresContainer.getUsername(),
                            "spring.datasource.password=" + postgresContainer.getPassword())
                    .applyTo(configurableApplicationContext.getEnvironment());
        }
    }
}
```

Joonis 22. Testcontainers konfiguratsioon

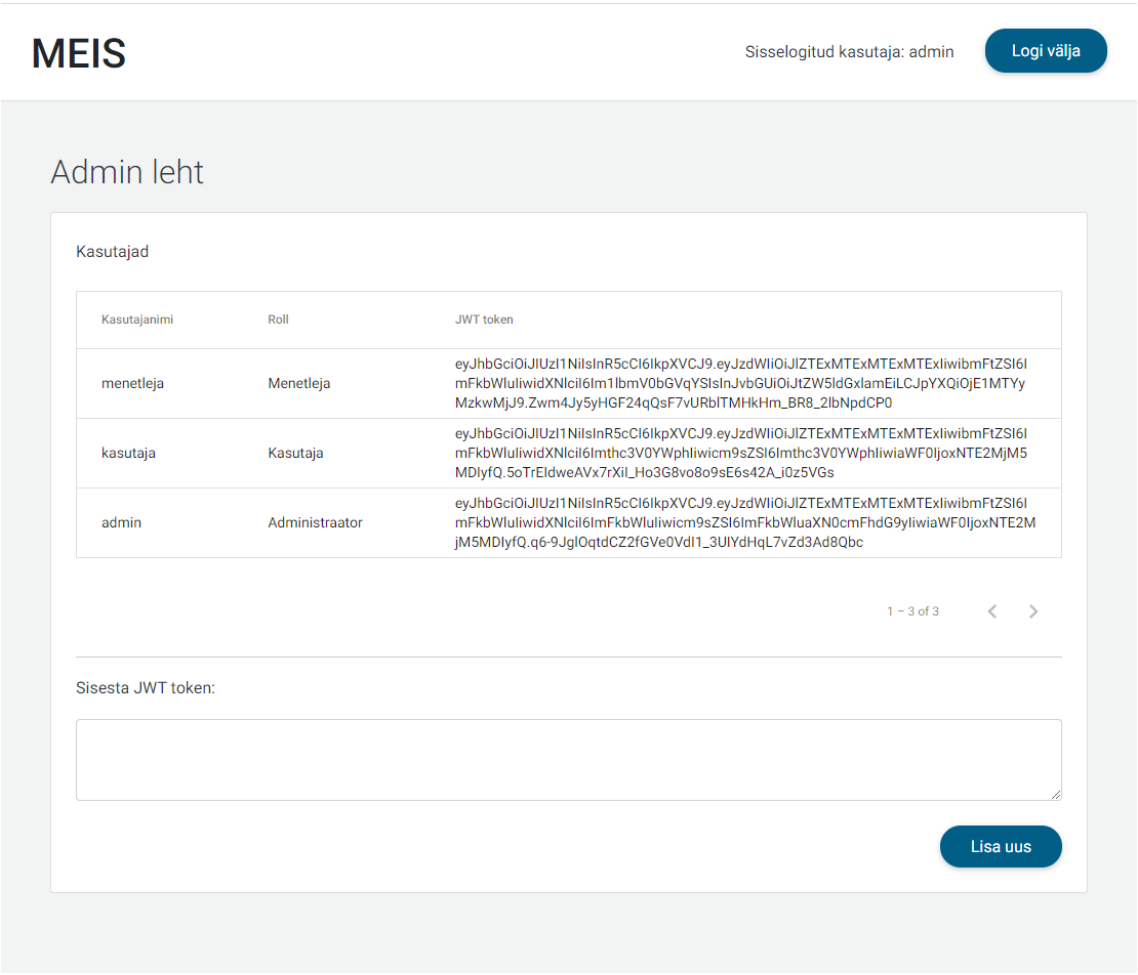
```java
@Test
public void testGetProcessingProceedings() {
    ProceedingRequest proceedingRequest = new ProceedingRequest();
    proceedingRequest.setUser(new UserDTO( username: "kasutaja", UserRoleEnum.USER.name()));
    ProceedingListResponse response = (ProceedingListResponse) rabbitTemplate.convertSendAndReceive(
            QueueConfig.GW_INTERNAL_EXCHANGE,  routingKey: "api.getProcessingProceedings", proceedingRequest);

    assertThat(response).isNotNull();
    assertThat(response.getPayload()).isNotNull();
    assertThat(response.getError()).isNull();
    assertThat(response.getPayload().size()).isEqualTo(1);
    assertThat(response.getPayload().get(0).getStatus()).isEqualTo(ProceedingStatusEnum.PROCESSING);
}
```

Joonis 23. Integratsioonitest menetluses sõnumite pärimiseks

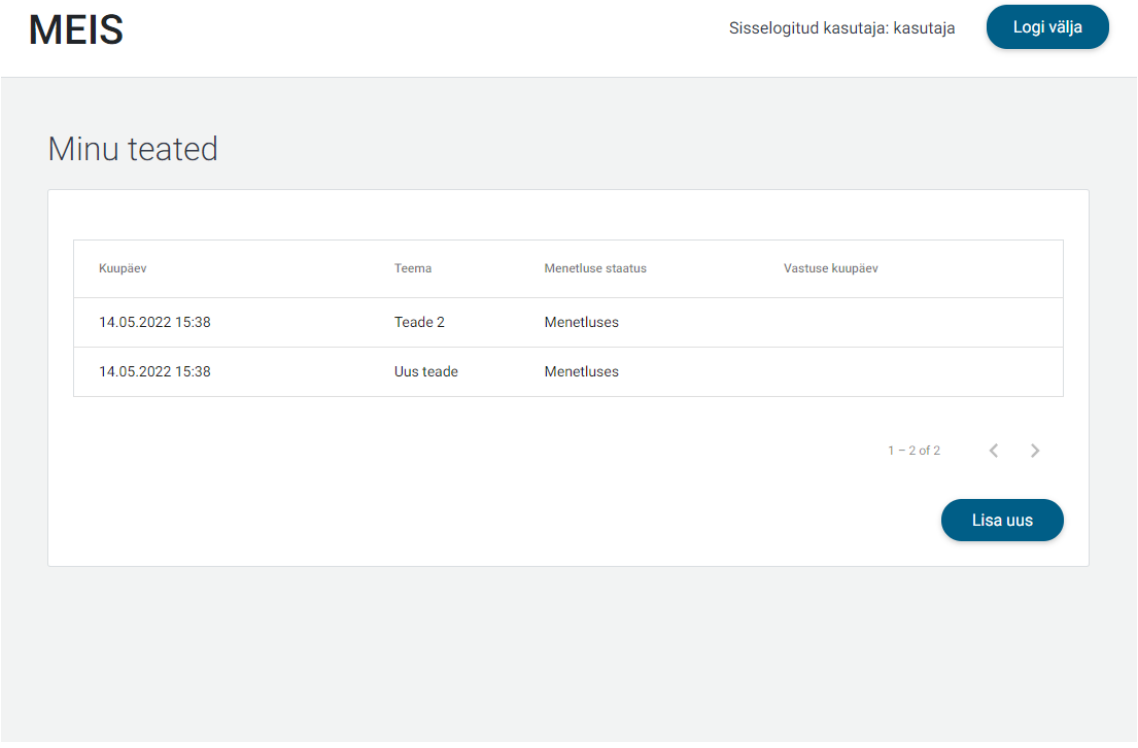
Lisa 11 – Kasutajaliides



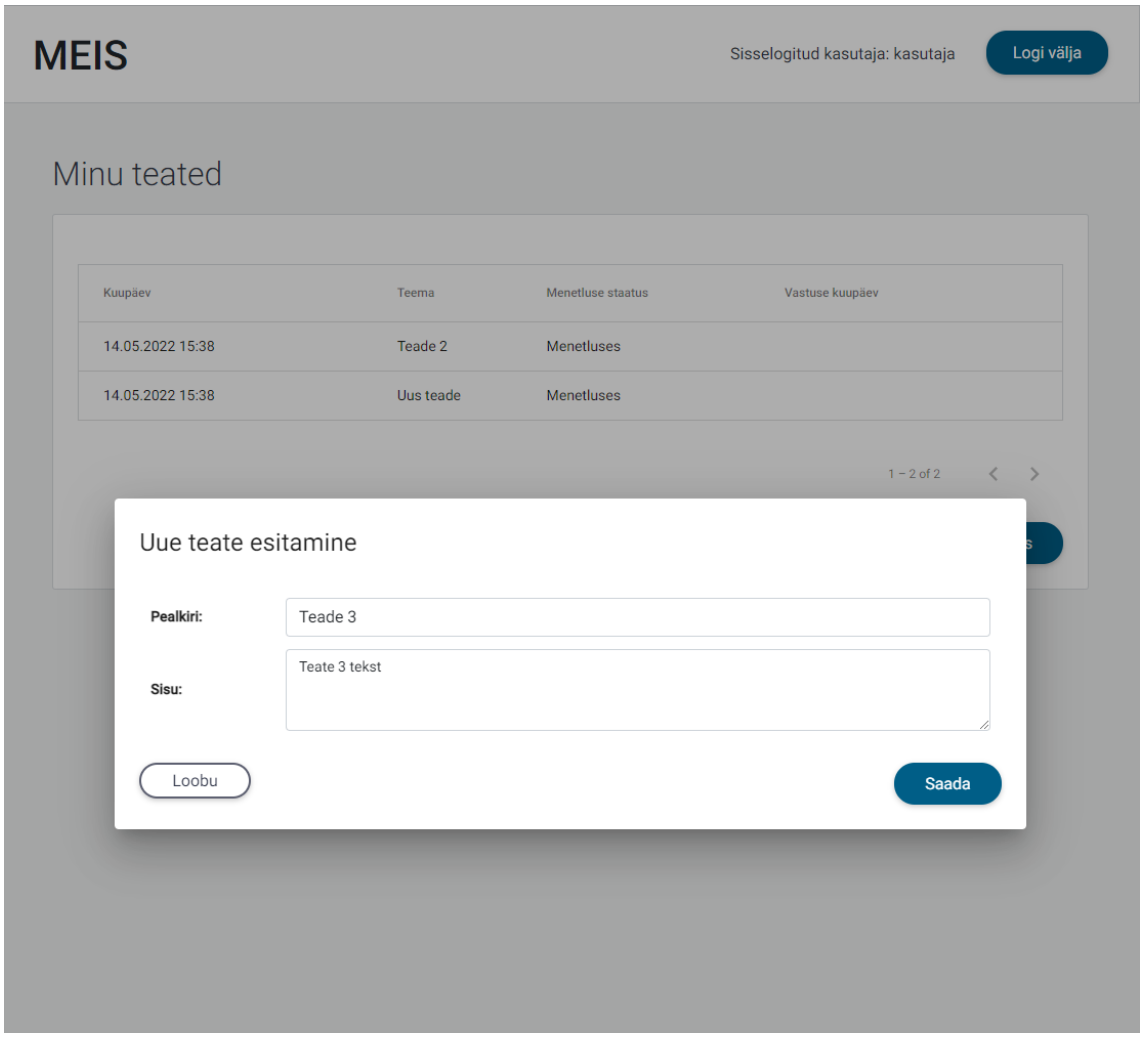
Joonis 24. Sisselogimisvaade



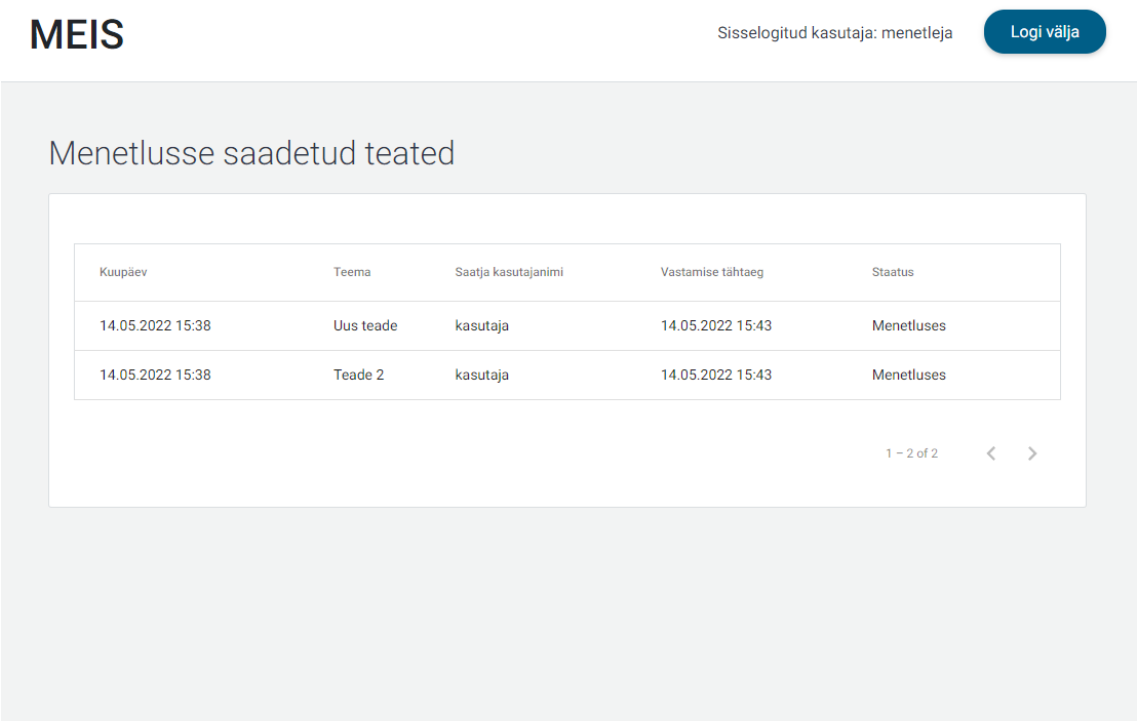
Joonis 25. Admin vaade



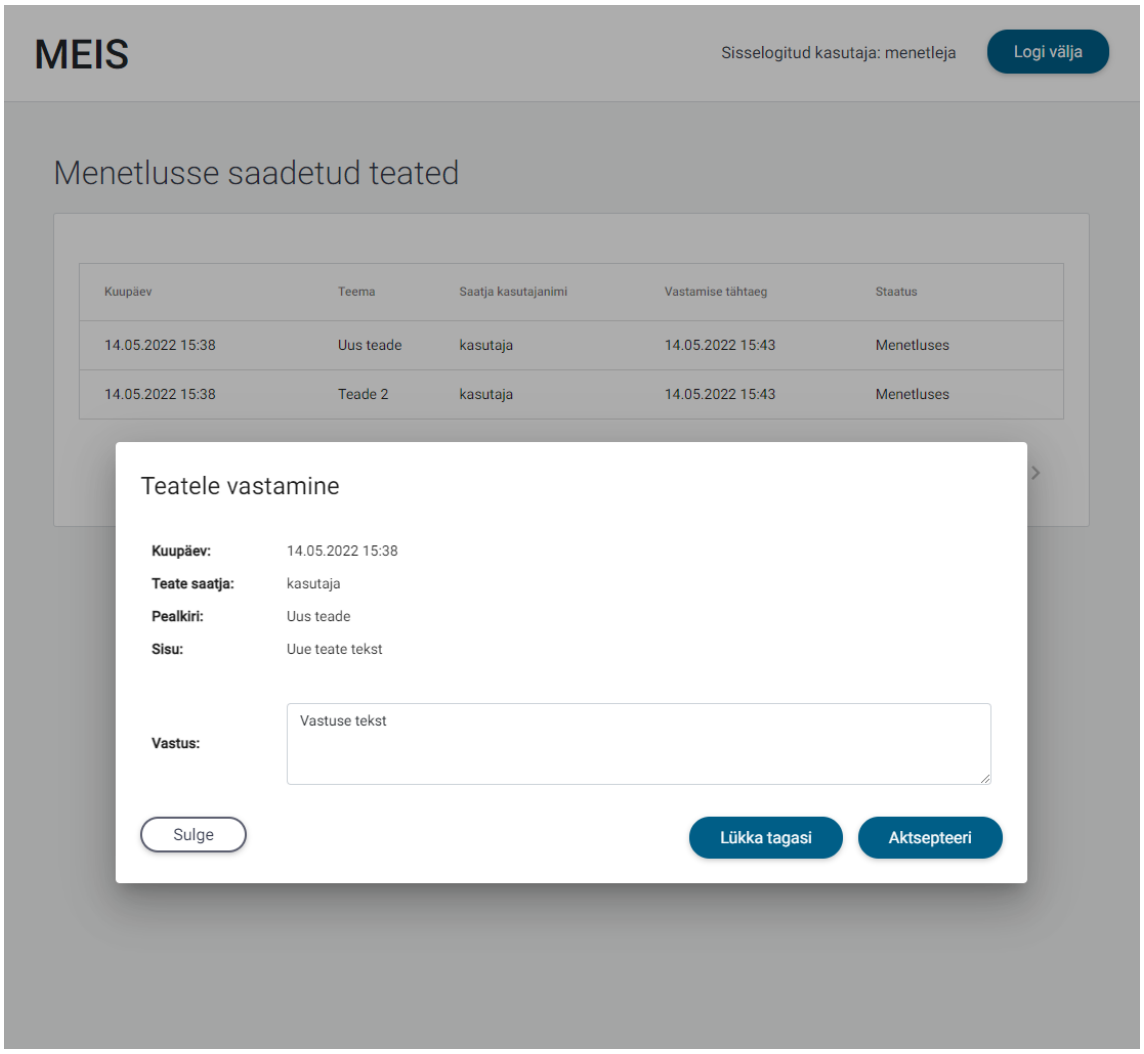
Joonis 26. Kasutaja vaade



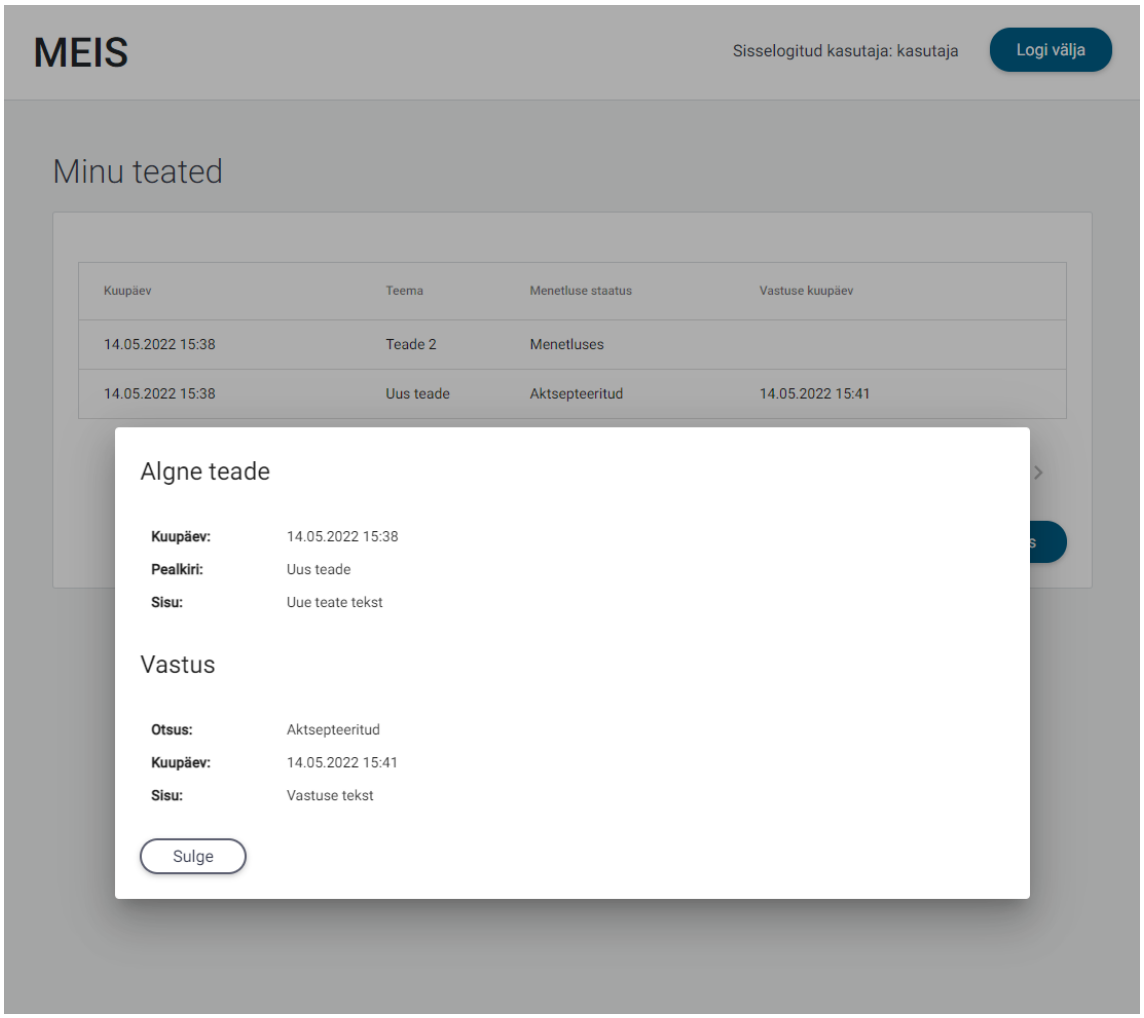
Joonis 27. Kasutaja uue teate esitamine



Joonis 28. Menetleja vaade



Joonis 29. Menetleja teatele vastamine



Joonis 30. Kasutaja teate lugemine

Lisa 12 – Java moodulite Gitlab CI/CD skriptid

```yaml
stages:
  - test
  - publish

variables:
  PUBLISH_DOCKER: "true"
  PUBLISH_SERVICE_LIB: "true"

include:
  - project: "industry62/dev-ops"
    file: "/gitlab/java/.gitlab-ci-java-init.yml"
  - project: "industry62/dev-ops"
    file: "/gitlab/java/.gitlab-ci-java-test.yml"
  - project: "industry62/dev-ops"
    file: "/gitlab/java/.gitlab-ci-java-publish.yml"
```

Joonis 31. Kasutajate mooduli töövoo skript

```yaml
variables:
  GIT_SUBMODULE_STRATEGY: recursive
  GRADLE_OPTS: "-Dorg.gradle.daemon=false"

image: gradle:jdk11

before_script:
  - export GRADLE_USER_HOME=`pwd`/.gradle
  - export GRADLE_VERSION=$(grep -E "^version" build.gradle | awk '{print $2}' | sed -e s/[\"\']//g)
  - export VERSION="$GRADLE_VERSION-$CI_PIPELINE_ID"
  - echo $VERSION
  - chmod 755 gradlew
```

Joonis 32. Java moodulite töövoole eelnevad tegevused

```yaml
test:
  stage: test
  services:
    - name: docker:dind
  variables:
    DOCKER_HOST: "tcp://docker:2375"
    DOCKER_TLS_CERTDIR: ""
    DOCKER_DRIVER: overlay2
  tags:
    - meis-proov
  script:
    - ./gradlew check
```

Joonis 33. Java moodulite testimise etapp

```
variables:
  PUBLIH_DOCKER: "false"
  PUBLISH_SERVICE_LIB: "false"

publish:
  stage: publish
  tags:
    - meis-proov
  script:
    - >
      if [ $PUBLISH_SERVICE_LIB != "false" ]; then
        echo "Publish service lib"
        ./gradlew publish
      fi
    - >
      if [ $PUBLISH_DOCKER != "false" ]; then
        echo "Publish docker"
        echo "./gradlew jib -Djib.to.image=$CI_REGISTRY_IMAGE:$VERSION"
        ./gradlew jib -Djib.to.image=$CI_REGISTRY_IMAGE:$VERSION -Djib.to.tags=latest-dev
        -Djib.to.auth.password=$CI_REGISTRY_PASSWORD -Djib.to.auth.username=$CI_REGISTRY_USER
        -Djib.httpTimeout=$JIB_HTTP_TIMEOUT
      fi
```

Joonis 34. Java moodulite konteineri ehitamise etapp

Lisa 13 – Kasutajaliidese CI/CD skript

```
FROM nginx:1.20.2-alpine
COPY dist/frontend /usr/share/nginx/html
EXPOSE 80
```

Joonis 35. Kasutajaliidese Dockerfile

```
image: docker:19.03.12

variables:
  DEV_DOCKER_IMAGE_TAG: DEV-$CI_PIPELINE_ID
  DOCKER_TLS_CERTDIR: "/certs"

stages:
  - build
  - publish

services:
  - name: docker:19.03.12-dind
    alias: docker

build:
  image: node:14.18.1-alpine
  stage: build
  tags:
    - meis-proov
  script:
    - echo "$TEHIK_NPM - $NPM_REGISTRY - $NPM_REGISTRY_AUTH"
    - >
      if [ "$TEHIK_NPM" == "true" ]; then
        echo -e "$NPM_REGISTRY_AUTH" > .npmrc
        echo "Set default registry to $NPM_REGISTRY"
        npm config set strict-ssl false
        npm config set registry $NPM_REGISTRY
      fi
    - npm install
    - npm link @angular/cli@13.0.3
    - npm run build
  artifacts:
    paths:
      - $CI_PROJECT_DIR/dist

publish:
  stage: publish
  tags:
    - meis-proov
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$DEV_DOCKER_IMAGE_TAG -t $CI_REGISTRY_IMAGE:latest-dev .
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker push $CI_REGISTRY_IMAGE:$DEV_DOCKER_IMAGE_TAG
    - docker push $CI_REGISTRY_IMAGE:latest-dev
```

Joonis 36. Kasutajaliidese Gitlab CI/CD töövoog